ISSN 0265-2919

90p 66

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication

IR£1.15 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.50

CONTENTS

APPLICATION

RESIDENT EXPERT Our series on artificial intelligence considers the development of expert systems — programs that incorporate highly specialised areas of knowledge — 1301

HARDWARE

COMEBACK BID Weighing in with a stylish design, large memory and seductive price, the Atari 130XE looks like it could have the makings of a champ — 1309

SOFTWARE

EXTENDED VIEW We take a look at how CP/M labels and manipulates its files — 1304

ICON TACT Beyond Software's Shadowfire uses Mac-like icons as part of a brain-teasing fast-action game — 1320

COMPUTER SCIENCE

SEARCHING LOOKS A discussion of PROLOG's unique tree-like search procedure — 1312

JARGON

FROM PET TO PLOTTER A weekly glossary of computing terms — 1308

PROGRAMMING PROJECTS

THE NEW WORLD I The first in a three-part complete listing of our New World simulation game. This week we provide flavours for the Spectrum — 1314

MACHINE CODE

CHANNELS OF THOUGHT We investigate how the Spectrum sends data to the screen and ZX printer via channels — 1317

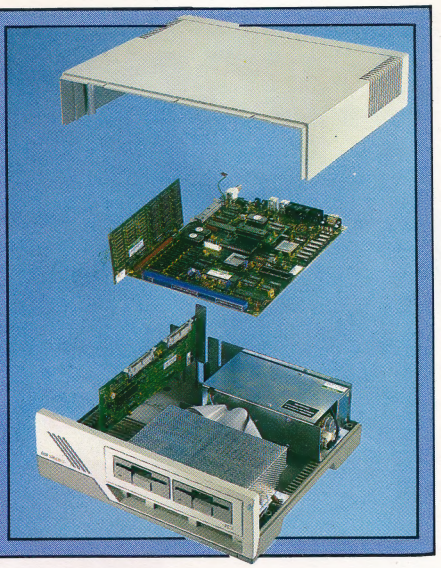
WORKSHOP

DIGITAL DOODLES Our digital tracer project concludes by adding some refinements to the control program we gave last week — 1306

REFERENCE CARD Another section of the Commodore 64's memory map — INSIDE BACK COVER

Next Week

- **The RM Nimbus has been greatly praised for its high performance. We take a look at Research Machine's latest 16-bit computer**
- **Our discussion of the Spectrum's operating systems continues with a further examination of the micro's control of channels and streams**
- **In our series on CP/M, we turn our attention to the manner in which files are organised on disk**



QUIZ

1) The Atari 130XE possesses one more chip than its predecessors. What is its name and function?
2) What is the general effect of a CALL command in Z80 Assembly language?
3) What is a 'wildcard'?

Answers To Last Week's Quiz

1) The Spectrum's printer buffer cannot be used to store machine code programs if a printer is being used or the program is over 256 bytes long.
2) If PROLOG fails to solve a problem through one chain of associations, it will 'backtrack' to a previous choice-point and attempt to obtain the answer by another route.
3) PIP stands for Peripheral Interchange Program and allows files to be copied under CP/M.
4) The function of a Peripheral Interface Adaptor (PIA) is to 'translate' the data between two incompatible interfaces.

IBM PC COURTESY OF COMPUTERLAND, 114 CHARING CROSS ROAD, LONDON WC2

Editor Stephen Cooke; **Art Editor** Claudia Zeff; **Production Editor** Bobby Pickering; **Deputy Editor** Steve Colwill; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Mike Clowes; **Sub Editor** Jon Kaye; **Contributors** Richard Forsyth, Graham Storrs, Geoff Bains, Joe Pritchard, Steve Malone, Anthony Ginn, Steve Colwill; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Alastair Gourlay; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
UK/EIRE – Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Price: AusS2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Price: NZS2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington.
ADDRESS FOR BINDERS AND BACK ISSUES – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
NOTE – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
ADDRESS FOR SUBSCRIPTIONS – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited. Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH

RESIDENT EXPERT

This instalment of our series on artificial intelligence focuses on expert systems — highly structured programs capable of fulfilling much of the diagnostic and advisory role of professional human practitioners in specialised areas of knowledge. We isolate the essential ingredients of the true expert system.

A human expert — a hospital consultant, a geologist, or a chemical analyst, for example — is usually someone we respect. Experts spend a long time studying and practising their skills to enable them to do their jobs well. The trouble with human experts, however, is that they are scarce, they are not always reliable, they want payment, and in the long run they die, taking much of their expertise with them. Many people are therefore keen on the idea of encoding expertise in computer programs, to get the benefits of being able to draw on a large body of knowledge without the drawbacks associated with the human experts.

The concept of the 'expert system' arose in the 1970s, when artificial intelligence (AI) researchers abandoned, or postponed, the quest for generally intelligent machines and turned instead to the solution of narrowly focused real-world problems. Thus the expert system is one of the first examples of applied AI, and expert systems techniques have spread out far beyond the confines of the research laboratories in which they were devised. Indeed, expert systems have to an extent brought AI into practical everyday use. Systems already exist that out-perform skilled humans at medical diagnosis, mass-spectrogram interpretation, classifying crop disease and much else besides. It is worth asking, therefore, how they work.

WHAT IS AN EXPERT SYSTEM?

Typically an expert system is based on an extensive body of knowledge about a specific problem area. In general, this knowledge will be organised as a collection of rules that allow the system to draw conclusions from given data or premises, thereby enabling it to offer intelligent advice or take intelligent decisions. This knowledge-based approach to systems design represents an evolutionary change within computer science, with revolutionary consequences. It replaces the traditional formula of 'Data + Algorithm = Program' with a new architecture centred around a 'knowledge base' and an 'inference engine', so that 'Knowledge + Inference = Expert System'. This formula is obviously similar, but different enough in approach to have profound implications.

MARCUS WILSON-SMITH

Seeking Professional Advice
Systems that can acquire knowledge from experts and use that knowledge to proffer advice or diagnoses are becoming increasingly popular in many disciplines — from medicine to farming and architecture. Expert systems, designed to operate within a narrow subject area, provide professionals with an on-line consultancy service to assist them in their work

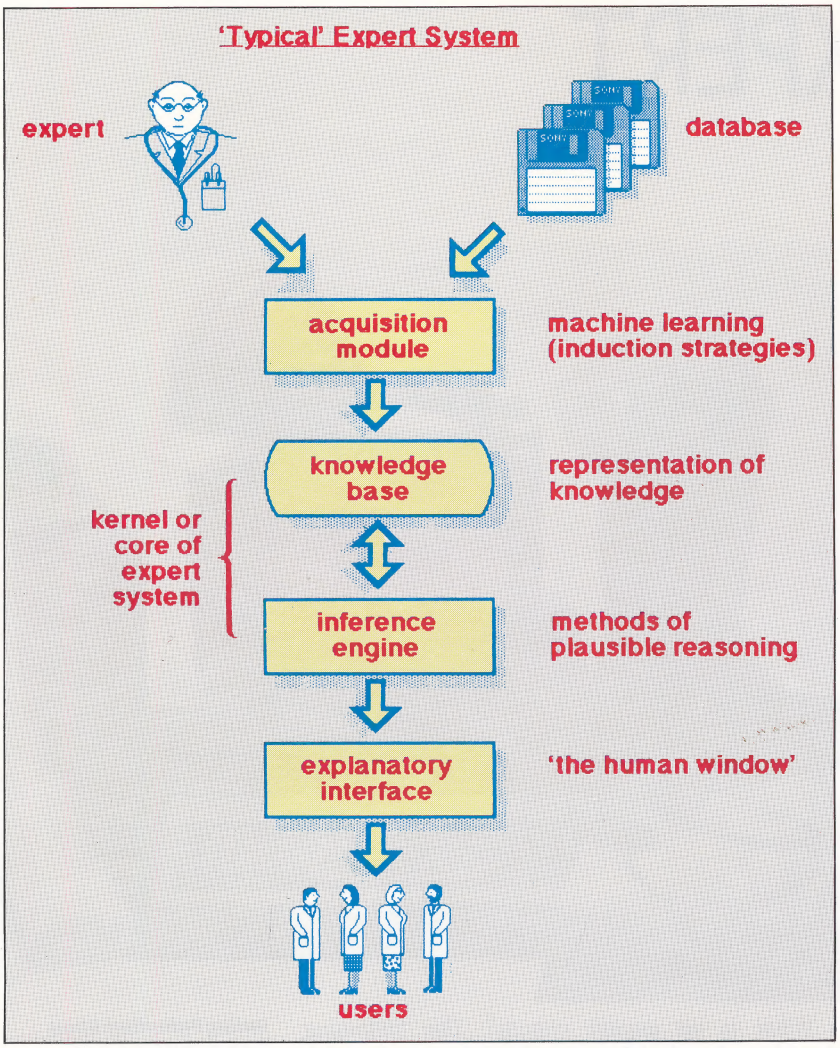
What is an expert system? The following checklist of typical features should prove helpful.

- An expert system is limited to a relatively narrow domain of expertise.
- It should be able to reason with uncertain data and unreliable rules.
- It must be able to explain its train of reasoning in a comprehensive way.
- Facts and inference mechanism are 'detachable': knowledge is not 'hard-coded' into the deductive procedures.
- It is designed to grow incrementally.
- It is typically rule-based.
- It delivers advice as its output — not tables of figures or graphs.

The key word is knowledge. Clearly the objective of an intelligent problem-solving system is to cut out blind or random search. To do so a computer system has to exploit the same advantage that the human expert has over the novice — that is, expertise, or organised knowledge: knowledge about facts, about rules of inference and about solution strategies. There are four essential components of a fully fledged expert system:

1. The knowledge base
2. The inference engine
3. The knowledge-acquisition module
4. The explanatory interface

IAN McKINNELL ON THE MACINTOSH



System Synopsis
An expert system comprises several modules that allow knowledge to pass from the expert to the end user. Knowledge must first be acquired from the expert(s) and incorporated into a knowledge base. In order to make predictions, give advice or provide a diagnosis, the system must then be able to draw inferences from the knowledge base. Finally, the explanatory interface allows the user to communicate with the system in order to consult it

All four modules are critical. A knowledge-based system may lack one or other of them, but a truly expert system should not. We will consider each of these four modules in turn and explain how they work together.

THE KNOWLEDGE BASE

The two fundamental components of an expert system are the knowledge base and the inference engine. The knowledge base stores information about the subject domain; however, information in a knowledge base is not the passive set of records and items that you would find in a conventional database. Instead it contains symbolic representations of experts' rules of judgment and experience in a form that enables the inference engine to perform logical deductions from it.

Most of the items in a knowledge base are non-mathematical. The two chief difficulties in developing a knowledge base are knowledge representation and knowledge acquisition. The former problem concerns the decision on how to encode knowledge so that the computer can use it. In general the following elements must be represented: domain terms (the jargon used by experts in the field); structural relationships (the interconnections of component entities); and causal relationships (the cause-effect relations between components).

The task of the knowledge engineer is to select appropriate means of storing such information symbolically. Four main methods have evolved:

- Rules in IF. . .THEN format. The condition specifies some pattern and the conclusion may be an action or assertion.
- Semantic nets. These represent relations among objects in the domain (e.g. the whale is a mammal) by links between nodes.
- Frames. These are generalised record structures which may have default values and may have actions coded as the values of certain fields or slots.
- Horn clauses. This is a form of predicate logic on which PROLOG is based and with which the PROLOG system can perform inferences (see page 1272).

Early expert systems used the rule-based formalism almost exclusively. A sample rule from the Mycin system for diagnosing blood infections is typical of the IF. . .THEN structure:

IF:
1. The infection requiring therapy is meningitis, and
2. The type of infection is fungal, and
3. Organisms were not seen on the stain of the culture, and
4. The patient is not a compromised host, and
5. The patient has been in a region where coccidiomycoses are endemic, and
6. The race of the patient is black or Asian or Indian, and
7. The cryptococcal antigen in the csf was not positive

THEN
There is suggestive evidence that cryptococcus is not one of the organisms which might be causing the infection.

From this example we can see that an expert system uses the technical jargon of the area in which it is designed to operate — in this case medicine. The IF. . .THEN construct used by Mycin is essentially a series of statements that can be determined as true or false. Thus, the statements can be linked by Boolean operators, such as AND, to assist computer manipulation. In order to elicit the information required to make a diagnosis, Mycin must enter into a dialogue with the system user. Obviously, in this system at least, the user must have a certain level of knowledge in the subject area, so that the expert system's queries can be understood and answered.

THE INFERENCE ENGINE

The inference mechanisms consist of search and reasoning methods that enable the system to find solutions and, if necessary, provide justifications for its answers. There are two overall reasoning strategies — forward chaining and backward chaining.

Forward chaining involves working forwards from the evidence (or symptoms) to the conclusions (or diagnoses). In a rule-based system

it simply involves matching the IF conditions to the facts, possibly in a predetermined order. Forward chaining is easy to computerise and is suitable in cases where all the data is to be gathered anyway. Examples of such cases are where the data is generated automatically by an instrument and where a form has to be filled in.

Backward chaining works from hypothesis to evidence. The system chooses a hypothesis and looks for data to support or refute it. It can be programmed in a recursive manner and in consultation-style systems typically leads to a more natural kind of dialogue. The problem of which hypothesis to pick in any given situation is not yet fully solved and so in practice most systems use a mixture of forward and backward chaining.

THE ACQUISITION MODULE

Experts are notoriously bad at saying how they reach their conclusions, not necessarily because they wish to preserve trade secrets but because many of their thought processes lie buried beneath the level of consciousness, at the intuitive level. So knowledge acquisition has come to be regarded as the main bottleneck in expert systems development. Experts tend to be good at criticism, however. They can look at an example case and say what decision should have been taken and, if required, criticise a computer's suggested solution. A good deal of attention has therefore been devoted recently to developing software tools that allow an expert system to induce its own knowledge from pre-classified examples. Effectively, this power assists the knowledge acquisition process. Overcoming at the same time many of the difficulties of extracting knowledge from human experts and the laborious task of coding it for the computer. Even if the system cannot do the whole job — from a database of instances to a set of decision rules — on its own, it may be useful if it can refine its own knowledge base during a period of 'apprenticeship' or during use, learning from its mistakes. We will take a closer look at some machine learning systems in forthcoming instalments.

THE EXPLANATORY INTERFACE

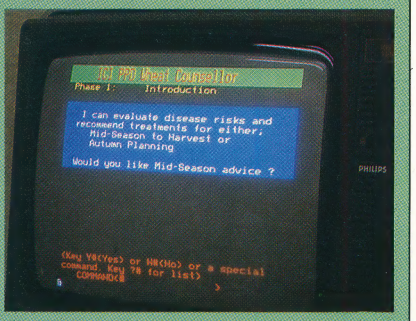
One of the good things about Mycin, Prospector and many classic expert systems was that they could justify their conclusions on request. Obviously when the responsibility for life-and-death or even profit-and-loss decisions is being handed over to the computer it is essential for the system to explain its reasoning. Here is an example of how Mycin replies to a WHY? question posed by its user (who is a physician, not a patient, and will not accept diagnoses on trust without explanation). The dialogue starts by Mycin requesting data.

Is meningitis a hospital-acquired infection?
WHY?
[2.0] This will aid in determining the organisms (other than those seen on cultures or smears) which might be causing the infection. It has already been determined that:
[2.1] the infection which requires therapy is meningitis, and
[2.2] organisms were not seen on the stain of this pending csf culture, and
[2.3] the type of infection is bacterial, and
[2.4] neurosurgery is not one of the relevant items from the history of this patient, and
[2.5] the age of this patient is greater than 10 days.
Therefore, if:
[2.6] the infection was acquired while the patient was hospitalised,
then there is evidence that the organisms (other than those seen on cultures and smears) which might be causing the infection are e.coli (0.75), staphylococcus-coag-pos (0.3), pseudomonas-aeruginosa (0.3), klebsiella-pneumoniae (0.5)
— from [RULE 545]

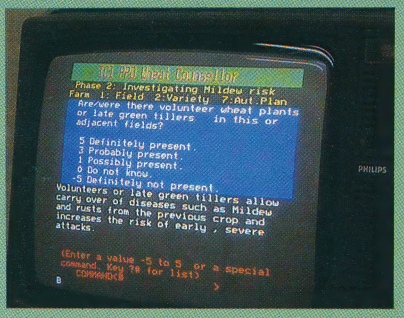
This explanation facility is in effect a partial trace through the program's reasoning process, couched in medical-jargon English. Such explanations can be produced fairly easily and cheaply in rule-based systems (by regurgitating the rules being used and their predecessors), which is a point in favour of the rule-based encoding of knowledge. Notice that the conclusions have numerical weightings attached. These are in fact not true probabilities. They are weightings that allow the system to deal with uncertainty in a consistent manner, and come up with an ordered list of likely diagnoses in the final analysis.

Be Well Advised . . .

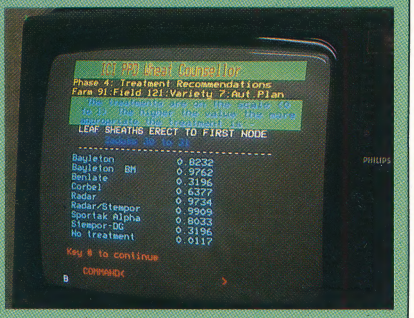
Wheat Counsellor can be entered via the Prestel system . . .


ICI PP1 Wheat Counsellor
Phase 1: Introduction
I can evaluate disease risks and recommend treatments for either, Mid-Season to Harvest or Autumn Planning
Would you like Mid-Season advice ?
PHILIPS

The initial introduction allows the user to select between mid-season and autumn planning advice . . .


ICI PP1 Wheat Counsellor
Phase 2: Investigating Mildew risk
Farm 1: Field 2:Variety 7:Aut Plan
Are/were there volunteer wheat plants or late green tillers in this or adjacent fields?
5 Definitely present.
3 Probably present.
1 Possibly present.
0 Do not know.
-5 Definitely not present
Volunteers or late green tillers allow carry over of diseases such as Mildew and rusts from the previous crop and increases the risk of early , severe attacks.
PHILIPS

The system then asks the farmer a series of questions on which it will base its advice . . .


ICI PP1 Wheat Counsellor
Phase 4: Treatment Recommendations
Farm 91:Field 121:Variety 7:Aut Plan
LEAF SHEATHS ERECT TO FIRST NODE
Key 0 to continue
PHILIPS

Having interpreted the farmer's responses, the Wheat Counsellor dispenses advice in the form of rating values for the various chemical treatments available

COURTESY OF ICI PLANT PROTECTION UK DIVISION

The Grain Brain

Wheat Counsellor is an expert system developed by ICI and available to farmers, free of charge, on Prestel. The system is designed to advise farmers on methods to combat wheat diseases, the correct chemicals to apply and an estimate of the likely losses. Although the system is designed to simulate the type of conversation that might be held over a farm gate, its knowledge has been accumulated from scientific studies of the spread of crop diseases. In order to gain the information from which it will draw its conclusions, the system leads the farmer through a series of simple questions. Its advice is then given in the form of a list of the treatments available, together with a suitability rating for each treatment.

EXTENDED VIEW

So far in our series on CP/M we have concentrated primarily on looking at the command structure of the operating system. We turn our attention now to the files upon which CP/M commands operate and how they can be manipulated within the system.

We have already discussed file extensions in CP/M, but let's now look at them in greater detail. As we have seen, a CP/M file consists of a primary name, which can contain up to eight characters, followed by a full stop and up to three letters, constituting a file extension (although this may be omitted). You can theoretically add any extension onto the primary filename, but some of these are reserved for specific purposes. For example, the COM extension is reserved for CP/M COMmand files. This informs CP/M that a file with this extension is to be added to the list of transient commands that can be run under it.

Similarly, BASIC programs written under CP/M should be stored with the file extension .BAS. Many versions of BASIC that run under CP/M will automatically assign this extension to a BASIC file, eliminating any need for you to type it in. Although the BASIC program will be stored as a source (text) file, the difference between this file and an ordinary text file is that when the program is reloaded prior to being run, CP/M will tokenise the BASIC program rather than list it as an ASCII sequential file. However, many machines will compile their BASIC programs before running them. Files that are compiled before execution will be stored with an .INT extension. This means that they are INTermediary files consisting of object code.

Machine code programs, like BASIC programs, can also have several different extensions assigned to them depending on their status. When we write a program in Assembly language, the source listing should have the extension .ASM attached. If the program does not have the correct extension, the resident assembler will not attempt to assemble the program and will simply generate a file error.

When we attempt to assemble the program, we must select one of two extensions. These are either .HEX or .PRN. The HEX extension means that what is produced at the end of assembly will be a HEXadecimal object file. Alternatively, with PRN the assembler will also produce a printed listing of the assembly with a copy of the source listing, the object code, the addresses of each of the op-codes and the list of assignments along with a list of errors. Thus, the PRN extension provides an essential aid in debugging assembly programs.

The final group of file extensions we shall look at are those associated with text files. Many versions of CP/M have a small text editing facility (called by executing the command file ED) provided within the operating system itself, whereas others assume you will have a specialised word processing package that can be run under CP/M. However, regardless of which you may be using, the file extensions remain the same.

Apart from the reserved filename extensions that have already been mentioned, almost any three letters can be used as an extension. In fact, you do not have to add an extension at all. If a text file is SAVEd without an extension having been added, CP/M will often add its own. For example, the TEXTED utility will add the extension .TXT on the end of a file whereas WordStar will add .$$$.

BACKUP FILES

If you look at the directory when a text file has been SAVEd you will notice that two files have been created: one with the file extension you have added and one with the suffix .BAK. This is a built-in safety feature of CP/M. Obviously, it is vital that important documents are not accidentally deleted or corrupted, so to prevent such disastrous occurrences CP/M will create two copies of all text files: a normal file and a backup .BAK file.

Thus, if you accidentally ERAse a file when editing, the backup version will always be available so that your previous work will not be irretrievably lost. When a file has been edited and then reSAVEd, a new version of .BAK will also be created. It is therefore a good idea to regularly reSAVE files so that the backup copy will be kept up to date.

Despite the fact that it is not necessary to add extensions, once implemented they are a useful aid in organising your files; not only for your own benefit — for displaying information about the organisation of files and headings — but also for the manipulation of groups of files. One of the most useful of these manipulations involves the use of 'wildcard' characters, which enable 'fuzzy matching' of files.

Suppose, for example, that a managing director has written a number of memoranda in the month of July. Sometime later he may decide these files are out of date and no longer needed in his catalogue. To delete July's

CP/M Control Characters

CP/M contains a number of control characters to perform many of its functions. Although several of these are now obsolete and seldom implemented, they are retained within the operating system standard to maintain compatibility between different computers

CTRL C	Reboots the CP/M system disk
CTRL C	Initialises newly inserted disk
CTRL M	Ends PIP commands and returns control to CP/M
CTRL Z	Returns control to ED after insert operation
CTRL P	Initially echoes all input to the printer. Retyping CTRL P exits this operation
CTRL U	Deletes current line
CTRL X	Deletes current line and moves cursor back to beginning
CTRL M	Executes the current command line (used instead of RETURN)
CTRL H	Backspace/delete
CTRL E	Allows a long command line to be input without executing at the end of a line
CTRL R	Repeats the current command line

memoranda, he could, of course, go through the catalogue and delete each of the files individually using the ERA command, but if there are a large number of files this could be a long and tedious process.

However, providing he has had the foresight to give all July's memos the same file extension, say .JUL, this process could be performed with a single command: ERA D:*.JUL (where D is the optional drive name). Placing the asterisk before the full stop tells CP/M to ignore the primary filename and ERAse the files with the .JUL extension. This asterisk can also be used on the right of the full stop. Using asterisks on either side of the full stop, the command will erase all the files on a disk, such as ERA*.*.

SELECTIVE FILE DELETION

Let's assume that the managing director keeps all July's files with the suffix .JUL. We shall now complicate matters by also assuming that there are a number of important documents that he does not want to delete. In this case it is out of the question to use the * prefix, since this would erase July's important files along with the redundant memos. However, our managing director has distinguished his memoranda from other files by naming them MEM1.JUL, MEM2.JUL and so on. However, it is still possible to delete all the memoranda with a single command without accidentally deleting other files, in which case executing the command ERA D:MEM?.JUL would be appropriate. Here, the operating system is being told to delete all files beginning with MEM and ending with the suffix .JUL. The ? in the fourth position signifies to CP/M that the fourth character is not significant and may be ignored.

This process can be adapted for any of the 11 possible positions in a file name. It is particularly useful should our managing director wish to delete memos for both JUNe and JULy, since both sets could be replaced by the format MEM?.JU?. Combinations of * and ? can appear in a file name so that ERA D:????Q???.* would erase all files with a Q in the fifth position of the prefix.

Use of the wildcard characters * and ? is not confined to the ERAse command but can also be used with, among others, the PIP, STAT and REN commands. Hence, in order to transfer a number of files from one disk to another, a typical command would be PIP B:=A*.*. This command will find all the files on drive A that match the format (which in this case would be all of them) and copy them onto the disk in drive B. In this way we can use wildcards to copy the entire contents of one disk to another.

Thus far in the series we have looked at commands that are used to control CP/M. However, the operating system also makes extensive use of control characters to perform many of its operations. Perhaps the most useful and common control character is the CTRL-C combination. Pressing these characters will produce a 'warm boot' to the system and reload CP/M into the computer. This is important, not just when reloading CP/M after a system crash, but also when inserting a new disk into one of the drives.

Remember that CP/M keeps a copy of a disk's 'log' in the computer's memory. When we exchange disks in a drive the operating system must be informed of the change, otherwise it may generate errors. When we warm boot CP/M, it will re-log the contents of each disk drive, which will enable us to continue. CTRL-C can also be used to halt the execution of many progams, since CP/M will automatically interrupt the system while it resets itself.

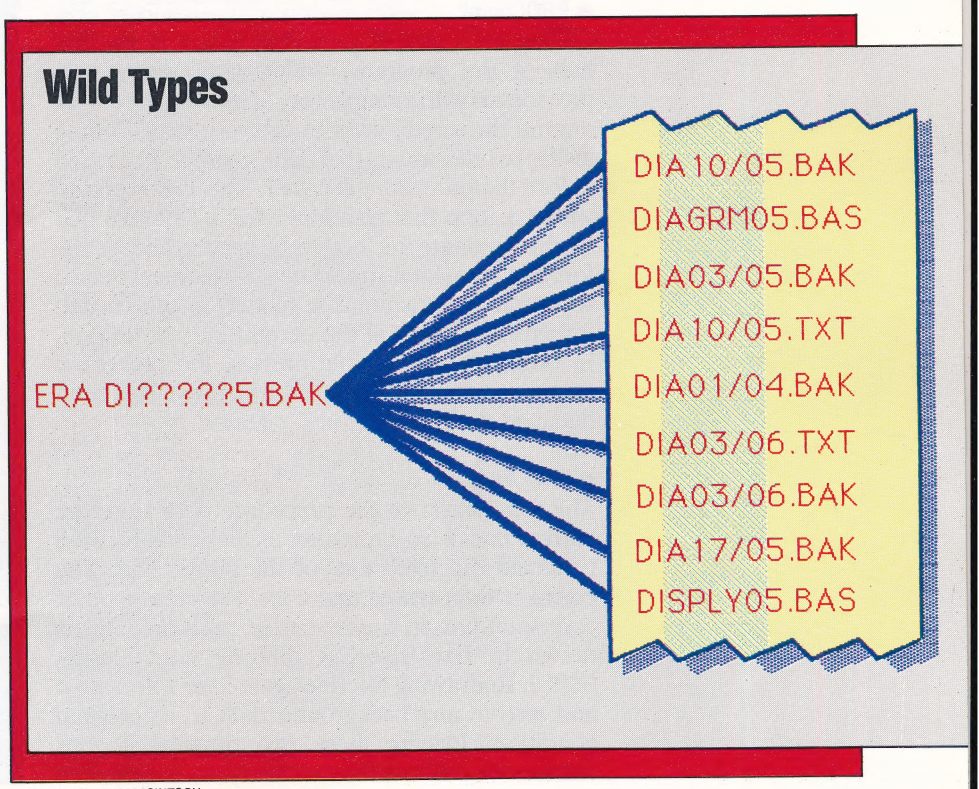
Many of the other control characters are concerned with editing text and have been adopted by a number of word processors that run under CP/M. For example, CTRL-H will delete the last character, whereas CTRL-U and CTRL-X will delete an entire line. If a printer is connected, text can be sent to it by typing CTRL-P. Pressing CTRL-P a second time halts this operation.

Other control characters are used in CP/M, many of which have been rendered obsolete by advances in hardware and software. There are several reasons why these characters have been retained even though their functions appear to have been duplicated by other keys; for example, most home computers now have a Delete key fitted. Part of the explanation is historical. Many early microcomputers did not have a Delete key or cursor keys, so that screen editing facilities had to be incorporated within the operating system.

Another reason for retaining the system is that of compatibility. While the CTRL and alphabetic characters have retained their ASCII equivalents over the years, the newer keys may not have, and so the control characters have remained to retain compatibility. Finally, a user who has spent several years with CP/M will have grown accustomed to using these keys. Altering the system now would mean that he would have to relearn control from the beginning.

Having finished our brief overview of the commands and files as implemented on CP/M, we can now carry out most of the everyday functions used within the operating system. In the next instalment, we'll delve deeper into CP/M to see how it is constructed and how the program interacts with the computer and its peripherals.

Walk On The Wild Side
The use of 'wildcard' characters is an important feature of CP/M. By entering 'wildcards' at different positions, the computer will effectively ignore any characters in that position when searching the directory. This means that by careful allocation of filenames the user can manipulate whole lists of files which are classified under different headings



LIZ DIXON ON THE MACINTOSH

DIGITAL DOODLES

We complete our digital tracer project by giving the second half of the tracer program for the BBC Micro. Adding this to the listing given in the previous instalment provides the user with an alternative method of enabling the tracer to produce points, lines, circles and filled triangles.

The second part of the tracer program controls the use of the tracer in an 'elastic mode'. Rather than simply fulfilling a freehand drawing function, the tracer can be used, when elastic mode is selected, to specify the start and end points of a line, the centre and radius of a circle, or the three corners of a filled triangle. Selection of each of these functions, together with a point-plotting fuction, is made by pressing the appropriate keys from the menu displayed at the top of the screen. Save, Load and Clear screen functions, similar to those for the freehand mode outlined in the previous instalment, can also be selected from this menu. The new section of program given here, therefore, allows complex patterns to be created simply on the screen, using the tracer and a few simple key-presses. These patterns can be saved to disk or tape to be reloaded later.

THE FOUR PROCEDURES

- PROCpoint

Calling PROCdraw (a procedure given in the first half of the program) with togflag set to one provides us with a simple way of moving the cursor around the screen without drawing lines. Setting togflag to one ensures that pen-up mode is selected before the procedure is called. This call is placed within a loop that also looks for a keypress that might terminate the loop or be used to change the foreground colour. In addition, if the tracer button is pressed then a point is plotted in the current foreground colour at the current cursor position. In order to stop double presses, the procedure cannot continue after a button press has been detected until the button is released.

- PROCline

The first action of the procedure is to store the current cursor co-ordinates, as these will be used to locate the fixed end of the elastic line. The cursor is then erased and a line drawn to a point corresponding to a new cursor position. This is drawn in Exclusive-OR plotting mode, using GCOL3. Redrawing this line again later will erase it and restore any background data to its original condition. Placing these two drawing actions within a loop allows the user to try out various line

Digital Tracer Commands

Main Menu

Select	Function
1	Freehand mode
2	Elastic mode

Freehand Menu

Select	Function
S	Save screen to tape or disk
L	Load screen from tape or disk
M	Return to main menu
C	Clear screen
Button	Pen up/pen down

Elastic Menu

Select	Function
S,M,L,C	As for freehand mode
P	Plot a point in current foreground colour. M returns to elastic menu
D	Draw line in current foreground colour from cursor position on entry to current cursor position. Press M or button to fix line and return to elastic menu
F	Fill a triangle between three points specified by button presses
R	Draw a circle in the current foreground colour. Centre is fixed by cursor position on entry and radius varied by current cursor position. Press M or button to draw the circle and return to the elastic menu

Foreground Colours

Select	Function
1	Selects logical colour 1 as foreground colour (default: red)
2	Selects logical colour 2 as foreground colour (default: yellow)
3	Selects logical colour 3 as foreground colour (default: white)

NB: Logical colours can be changed using the VDU 19 command to select from the 16 colours available. For example, VDU 19,1,6,0,0,0 changes logical colour 1 to cyan (actual colour 6).
Foreground colour can be reselected at any time

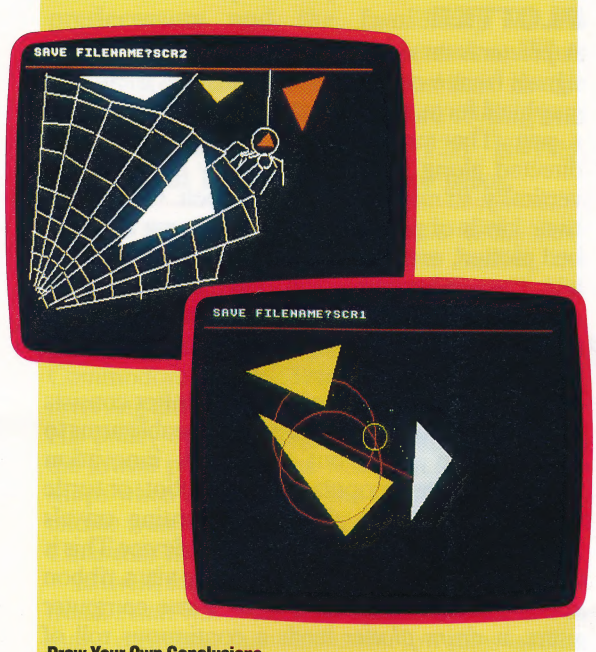
positions, as though the line were an elastic band attached at one end to a fixed point and at the other to the tracer's screen cursor. When satisfied with a line's position the user can 'fix' the line by pressing the tracer button or M on the keyboard. In this case the procedure omits to draw the line a second time, leaving it unerased, before returning to the main menu.

- PROCfill

The fill procedure takes advantage of the BBC's triangle-filling feature, accessed by using a PLOT 85,x,y command. This command takes the last two points visited and the point specified in the command as the three corners of a triangle and fills the shape with a solid block of colour. The procedure uses a similar method to that used in PROCpoint to allow the user to plot the three corners of the triangle to be filled, using the tracer button. The last two lines of the procedure revisit the three points in order to fill the triangle shape.

- PROCcircle

The final facility offered by the program allows the user to specify the radius of a circle using the same elastic band method as that used in PROCline. In fact, PROCline is called from within the circle-drawing procedure for exactly this purpose. Once the radius is selected control returns to PROCcircle, which must first erase the line left by PROCline and erase the cursor before drawing the circle. The circle's centre will be held in x1 and x2, the cursor co-ordinates on entry to the procedure, and PROCline provides the co-ordinates of a point on the circumference, (x2,y2). Using these two points, the circle's radius can be calculated and a standard circle-drawing algorithm used to create the circle.



Draw Your Own Conclusions
Adding the second part of the Digital Tracer program, given here, to the section given in the previous instalment allows the tracer to be used to draw straight lines, circles and filled triangles to create screen displays such as these with ease

Digital Tracer Program Part 2

The following listing forms the second part of the digital tracer program for the BBC Micro. It should be added to the first part of the listing given on page 1288

```
 2350 DEF PROCelastic
 2360 PROCelastic_inform
 2370 REPEAT
 2380 ans$=INKEY$(1):IF ans$<>"" THEN PROCelastic_
press
 2390 togflag=1:PROCdraw
 2400 UNTIL exitflag=1
 2410 ENDPROC
 2430 DEF PROCelastic_inform
 2440 PROCcalc_xy:oldx=x:oldy=y:PROCcursor(oldx,oldy)
 2450 PROCreinform
 2460 GCOL 0,1:MOVE 0,920:DRAW 1280,920
 2470 ENDPROC
 2480 DEF PROCreinform
 2490 PRINT TAB(1,1);SPC(79)
 2500 PRINT TAB(1,1);"S=Save L=Load M=Menu C=Clear
R=Circle"
 2510 PRINT" D=Draw Line    P=Point    F=Fill"
 2520 ENDPROC
 2540 DEF PROCelastic_press
 2550 IF ans$="C" THEN CLS:PROCelastic_inform:ENDP
ROC
 2560 IF ans$="M" THEN exitflag=1
 2570 PROCcolour_change
 2580 IF ans$="R" THEN PROCcircle_title:PROCcircle
:PROCelastic_inform:ENDPROC
 L.2590,2600
 2590 IF ans$="D" THEN PROCline_title:PROCline
 2600 IF ans$="P" THEN PROCpoint_title:PROCpoint
Celastic_inform:ENDPROC
 2620 IF ans$="S" THEN PROCsave_screen:PROCelastic
_inform:ENDPROC
 2630 IF ans$="L" THEN PROCload_screen:PROCelastic
_inform:ENDPROC
 2640 ENDPROC
 2660 DEF PROCpoint_title
 2670 PRINT TAB(1,1) SPC(79)
 2680 PRINT TAB(15,1);"Point Mode"
 2690 ENDPROC
 2710 DEF PROCpoint
 2720 REPEAT
 2730 togflag=1:PROCdraw
 2740 ans$=INKEY$(1)
 2750 PROCcolour_change
 2760 IF (ADVAL(0)AND 3)<>0 THEN PLOT 69,x,y:REPEA
T UNTIL (ADVAL(0)AND 3)=0
 2770 UNTIL ans$="M"
 2780 PROCreinform
 2790 ENDPROC
 2800 DEF PROCline_title
 2810 PRINT TAB(1,1) SPC(79)
 2820 PRINT TAB(15,1);"Line Mode"
 2830 ENDPROC
 2850 DEF PROCline
 2860 x1=x:y1=y:REPEAT
 2880 PROCcalc_xy:PROCcursor(oldx,oldy)
 2890 x2=x:y2=y:GCOL 3,colour:MOVE x1,y1:DRAW x2,y2
 2900 ans$=INKEY$(1)
 2910 IF (ADVAL(0)AND 3) THEN ans$="M"
 2920 PROCcursor(x,y):oldx=x:oldy=y
 2930 IF ans$<>"M"THEN GCOL 3,colour:MOVEx1,y1:DRA
W x2,y2
 2940 PROCcolour_change
 2950 UNTIL ans$="M"
 2960 PROCreinform
 2970 ENDPROC
 2990 DEF PROCfill_title
 3000 PRINT TAB(1,1) SPC(79)
 3010 PRINT TAB(15,1);"Fill Mode"
 3020 ENDPROC
 3040 DEF PROCfill
 3050 FOR I=1 TO 3
 3060 REPEAT
 3070 togflag=1:PROCdraw:ans$=INKEY$(1)
 3075 IF ans$<>""THEN PROCcolour_change
 3080 UNTIL (ADVAL(0)AND 3)<>0
 3090 PLOT 69,x,y:x(I)=x:y(I)=y
 3110 REPEAT UNTIL (ADVAL(0)AND3)=0:REM AWAIT BUTT
ON OFF
 3120 NEXT I
 3135 PROCcursor(x,y):REM cursor off
 3140 FOR I=1 TO 2:PLOT 69,x(I),y(I):NEXT I
 3150 PLOT 85,x,y
 3160 ENDPROC
 3180 DEF PROCcircle_title
 3190 PRINT TAB(1,1) SPC(79)
 3200 PRINT TAB(15,1);"CIRCLE MODE"
 3210 ENDPROC
 3230 DEF PROCcircle
 3240 PROCline
 3250 GCOL 3,colour:MOVE x1,y1:DRAW x2,y2:REM RUBO
UT  LINE
 3260 GCOL 0,colour:REM BACK TO NORMAL PLOT MODE
 3270 PROCcursor(x2,y2)
 3370 radius=SQR((x2-x1)^2+(y2-y1)^2)
 3380 MOVE x1+radius,y1
 3390 FOR angle=0 TO 2*PI STEP 0.1
 3400 rx=x1+radius*COS(angle)
 3410 ry=y1+radius*SIN(angle)
 3420 DRAW rx,ry
 3430 NEXT angle
 3435 DRAW x1+radius,y1
 3440 ENDPROC
```

PET

The Commodore *PET*, or Personal Electronic Transactor, represents one of the milestones in the development of the microcomputer. Its appearance in 1977 can be considered as one of the major events that directed computing beyond the realm of data processing departments and electronics enthusiasts towards the 'dream' of a computer in every home.

The precedent established by the PET was that it was not only fully assembled with its own casing, but it also had a built-in cassette deck and monitor, as well as BASIC resident in ROM. You had only to plug in the machine and power it up in order to use it. Previously, microcomputers were available only as a kit or with minimal assembly at best.

PICTURE PROCESSING

Also known as 'image processing', *picture processing* is the digital analysis of signals supplied from a television camera or similar device which, after processing, can be displayed either on a television or video monitor or as a printout. The amount of detail displayed is a function of the definition (resolution) of the image, of which each point is referred to as a pixel. So, the greater the number of pixels there are within the picture, the higher the definition is said to be.

The image received from the camera is held as a two-dimensional array within the computer. The individual pixels can be stored within the array in a number of different ways depending on the program used. At the simplest level, however, the information stored can merely state whether that pixel is on or off (whether or not light is falling on the corresponding point of the camera image). However, other information can be stored and manipulated within the computer, and this may specify colour or shades on the 'grey scale' (the relative contrast of the pixel compared with others in a monochrome image).

Picture processing is finding an increasing number of applications in the television and satellite communications fields. These range from the stunning effects now being produced in films and videos to improving the definition of the sometimes hazy pictures received from space probes. Picture processing is also being applied in the field of artificial intelligence where it is used with the emerging techniques of pattern recognition. In this area the emphasis is not merely on programming a computer to produce a required display, but rather enabling it to interpret an image and attempt to understand its 'meaning'.

PLASMA DISPLAY

The method of *plasma display*, which is increasingly being used in computers, is characterised by an orange or red light emanating from characters similar in design to those used on LCD displays. A plasma display is constructed by trapping a gas between two materials, one of which must be transparent (to enable you to see the display). At various points around the display, electrical lines are connected to form a matrix, so that when a current is applied to them, the gas will become charged and emit light. In this way, various combinations of currents through the lines can produce characters or designs.

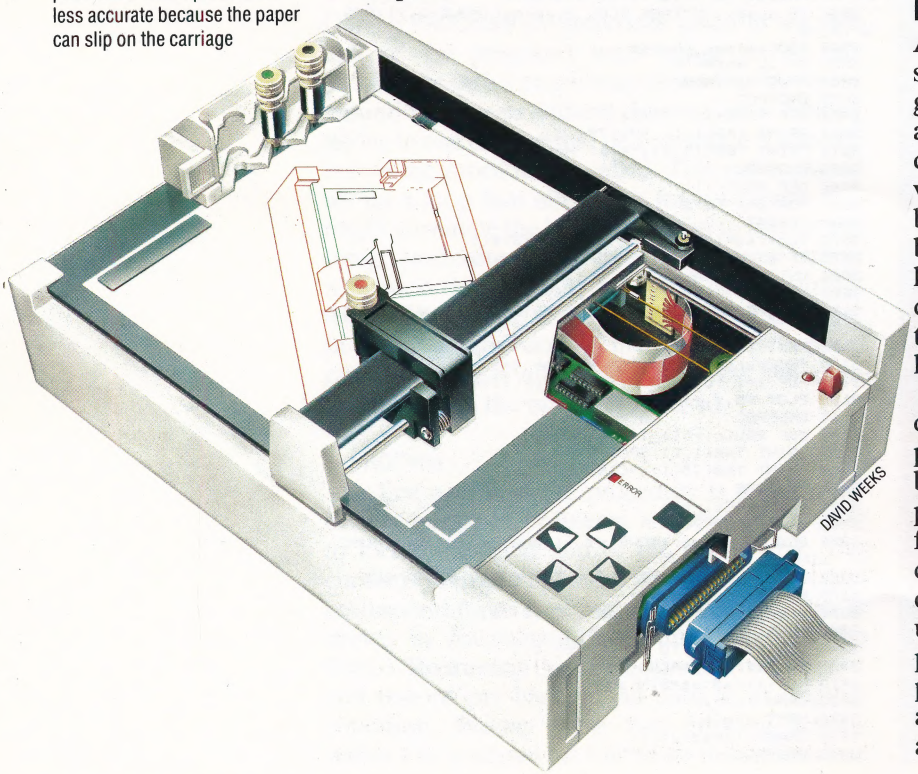
For the most part, plasma displays are made to produce single characters, although it is possible to build larger displays. However, the costs of these tend to be too high, especially since their functions can be performed just as well by devices incorporating cathode ray tube technology.

PLOTTER

A *plotter* is an output device that receives digital signals from a computer and translates them into graphical form, usually onto a piece of paper with a pen. The movements of the plotter can be directed along either the x and y axes, or else vertically, to raise and lower the pen. Of the two basic kinds of plotter, the first is known as a 'flat bed plotter', onto which a single piece of paper is laid. The pen is mounted on a bar that moves along one axis, while the pen itself moves along the bar, thus able to plot on any given point within the limits of the hardware.

The most usual type of flat bed plotter consists of a board to hold the paper and moveable arms to position the pen, but a recent innovation beginning to gain popularity is the robot, or turtle plotter. With these, the pen mounting travels freely over the paper on a motorised device. This is controlled by the computer, usually via a ribbon cable. Perhaps more familiar to home computer users, a *drum plotter* feeds a continuous roll of paper onto a drum, much like a typewriter. The pen mounting moves across the paper, along the x axis, while vertical movement, along the y axis, is achieved by the drum rolling the paper through.

The Plot Thickens
There are two kinds of printer plotters — drum and flat bed. Although drum plotters are more common, the more expensive flat bed plotters, like the one shown here, are better for precision work. This is because the paper is held steady and the pen can be accurately positioned. Drum plotters are less accurate because the paper can slip on the carriage



COMEBACK BID

CHRIS STEVENS

Getting Back Into Shape
The Atari 130XE is the first product to be launched under the company's current president, Jack Tramiel. With 128 Kbytes of RAM on board, at a price that compares favourably with many computers with only half the memory, Atari hopes that this will be the machine that will revive the company's fortunes

With its new 130XE micro Atari makes a virtue of compatibility both with its XL predecessors and the familiarity of its graphics, sound and other features. What places it in a challenging position in the market, however, is its seductive styling and RAM capacity of a full 128 Kbytes, available at a very competitive price.

Despite the fact that Atari was one of the early pacesetters in the home computer market, the early 1980s were not good years for the corporation. In 1984, after suffering heavy losses, it was taken over by the former head of Commodore, Jack Tramiel, who set about reversing the fortunes of the ailing computer giant. His policy of marketing computer technology at the lowest possible price was soon evident in the shops when the prices of the 600XL and 800XL micros were dramatically reduced in time for the Christmas market.

In itself, this move was not enough to reverse the trend. Atari was faced with a problem common to many computer manufacturers: low sales leading to lack of software support, which stifles sales even more — meaning that funds are lacking for further investment in new machines.

It is ironic that Jack Tramiel was the person most responsible for placing Atari in this position. His aggressive marketing of the Commodore 64 led to Atari being almost wiped out by its rival. By the beginning of 1985, the situation was changing dramatically. Commodore was faced with falling sales and the failure of its Plus-4 to make a major impact on the market, while Atari announced a host of new products. The first of the new machines is the Atari 130XE, a computer based around the 6502C processor.

The 130XE is essentially the same machine as the basic Atari eight-bit computer, which has been around in one form or another since the beginning of the 1980s. The main difference between this machine and earlier Atari computers is the sleek new styling and the massive amount of on-board memory: the Atari 130XE boasts a full 128 Kbytes of RAM.

The casing of the computer looks very different from that of its predecessors. The light grey plastic outer shell has the elegant design the public has come to expect of a modern computer, with rounded lines and wide, sculpted keys that allow for ease of typing. The keys have a slightly better travel than that offered by previous models, with the added advantage of not rattling as you type.

In common with the other 6502-based Atari micros, the 130XE has five pre-programmed function keys. However, unlike the previous

models, these have been placed above the main keyboard rather than on the right-hand side. They are now moulded in the same grey plastic as the rest of the casing and are designed in the stylish form of parallelograms rather than squares. These are a great improvement on those of the XL series, which were metal and had a distinctly unsure and wobbly feel.

INTERFACE CONNECTIONS

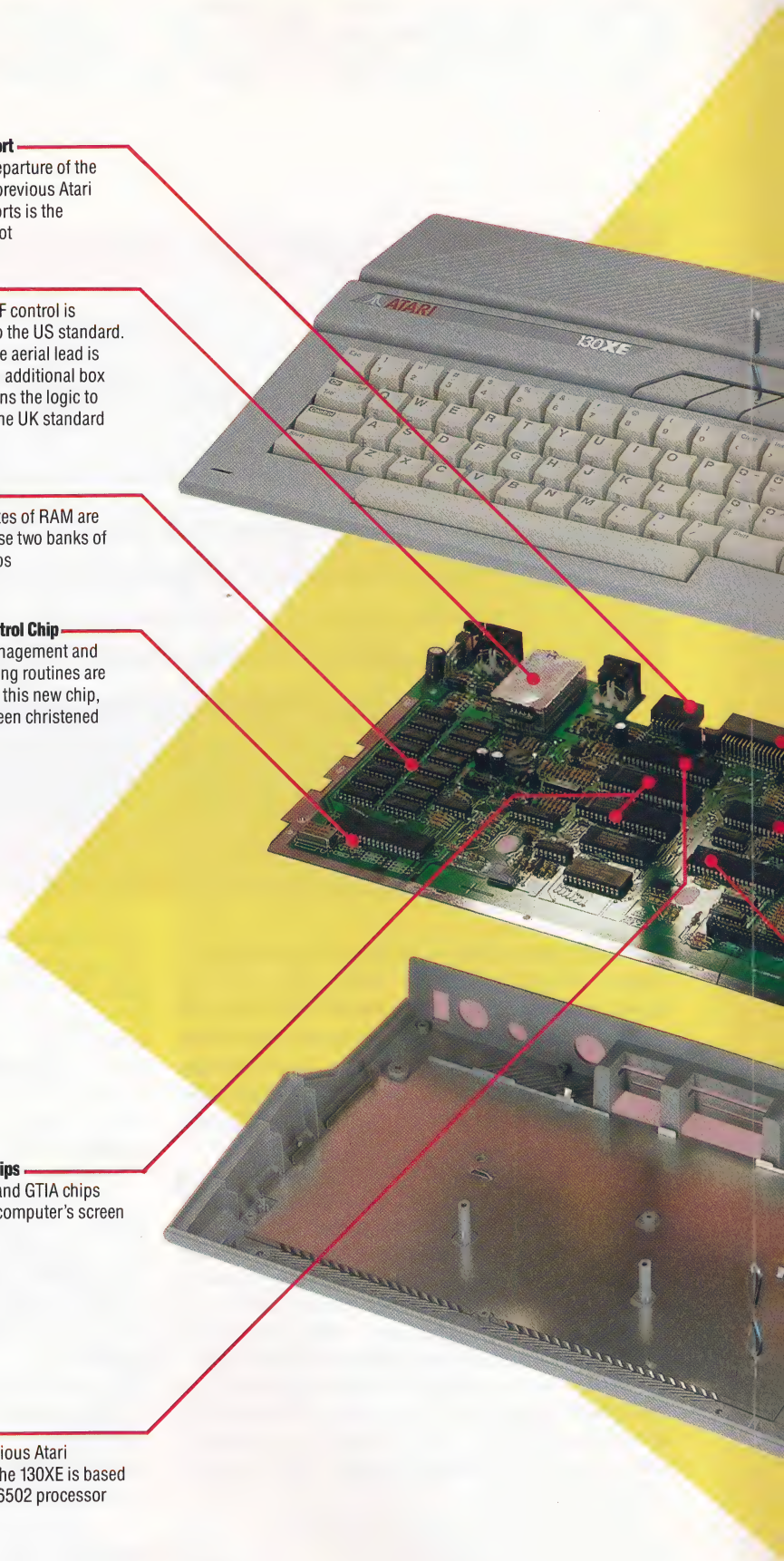
The interfaces fitted on the rear and right-hand side of the new Atari also hold few surprises. On the side are the expected nine-pin D-type joystick ports first used by Atari and since adopted by almost everyone else. At the back of the machine is the 13-pin serial control port that is used by Atari to fit and daisychain Atari peripherals such as cassette decks, disk drives and printers. To the right, set into the casing, are the cartridge and expansion interfaces. The cartridge port enables the machine to run the large quantity of highly rated Atarisoft games cartridges, such as Pacman and Galaxians, that have been one of the strengths of the company over the years.

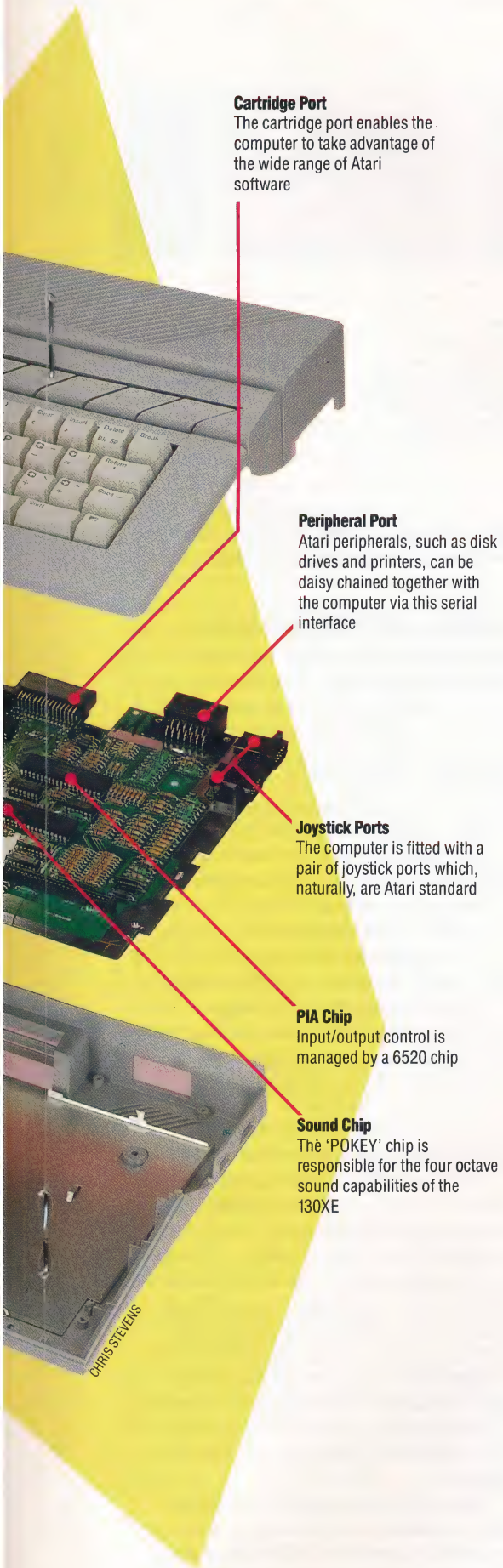
The expansion port, however, is something of a departure from the previous standard. Earlier Atari computers had a 50-way edge connector as an expansion bus. The new machine has a much smaller 14-way bus fitted as a cartridge connector. The remaining interfaces on the 130XE are the composite monitor socket, an RF jack plug for television sets and the standard Atari power-supply socket.

To keep costs down the 130XE has no on-board modulator to allow it to work effectively with British standard televisions, but has the US standard instead. Thus, to allow UK sets to be used, the aerial lead from the computer to the television runs via a small box that modifies the signal into one that will correctly process the television display.

Styling and compatibility are all very well, of course, but Atari applied these to the XL series without conspicuous success. What makes the 130XE different from earlier machines and what is its obvious selling point is the availability of a large memory capacity at such a low price. An eight-bit microprocessor can, of course, address only 64 Kbytes of RAM at a time. To address twice that amount the computer has to make use of the process known as 'bank switching'. Using this technique, the computer can look at a 'window' of 64 Kbytes among the total 128 Kbytes. While the technique is not perfect, and although the extra instructions necessary to switch from one bank of RAM to another result in somewhat slower retrieval speeds, it does allow an eight-bit micro to address more memory than would otherwise be possible.

Actually, the bank switching technique is more common than you might think. Both the Oric Atmos and the Commodore 64 have more than the normal complement of 64 Kbytes on board, and both use bank switching as a means of making


Expansion Port
The single departure of the 130XE from previous Atari peripheral ports is the expansion slot
RF Socket
The 130XE RF control is configured to the US standard. Therefore, the aerial lead is fitted with an additional box which contains the logic to conform to the UK standard
RAM Chips
The 128 Kbytes of RAM are stored in these two banks of 8- Kbyte chips
Memory Control Chip
Memory management and bank switching routines are contained in this new chip, which has been christened 'Freddy'
Graphics Chips
The ANTIC and GTIA chips control the computer's screen graphics
CPU
Like all previous Atari machines, the 130XE is based around the 6502 processor
ATARI
130XE


Cartridge Port
The cartridge port enables the computer to take advantage of the wide range of Atari software
Peripheral Port
Atari peripherals, such as disk drives and printers, can be daisy chained together with the computer via this serial interface
Joystick Ports
The computer is fitted with a pair of joystick ports which, naturally, are Atari standard
PIA Chip
Input/output control is managed by a 6520 chip
Sound Chip
The 'POKEY' chip is responsible for the four octave sound capabilities of the 130XE
CHRIS STEVENS

full use of the available memory.

What is unusual is the price of the 130XE. Although more expensive than the Sinclair Spectrum and Acorn Electron, the 130XE is considerably cheaper than the list prices of the BBC Micro and the Commodore 64. In order to provide a 128 Kbyte computer at that price and produce a profit, Atari needs to have cut production costs considerably. To an extent, the nature of the machine itself will have kept the price low: the 130XE is essentially a revamped machine, which means that development costs have been kept to a minimum. The main economies have taken place inside the machine.

CUTTING COSTS

The memory area comprises 16 eight-Kbyte RAM chips. The production cost of these chips, which are no longer considered the product of 'leading edge technology', has fallen dramatically in the past few years and this is reflected in the price. Another way of reducing costs is to keep the number of components on the board to a minimum. Although many of the chips on the earlier XL series have found their way into the 130XE in order to maintain compatibility, the printed circuit board is exceptionally well laid out, looking considerably less cluttered than many machines with only half the memory capacity. Finally, Atari has invested heavily in automated assembly plants, of which the 130XE is the first product. All the components on the board are soldered on by machine.

As none of the sound, graphics and BASIC ROM chips have been altered in essentials, the computer is, for the user, exactly the same as the earlier models, with the high-performance sound and graphics with which Atari has become associated. One major change of benefit to the user is the manual. The handbooks that came with earlier models tended to be oversimplified to the point where they appeared to be aimed at children. The BASIC tutorial is much improved and the company has given some technical specifications in the appendix. However, for a full explanation of the dialect, one still needs to purchase the Atari BASIC Reference Manual.

Although the Atari range of micros was sorely in need of upgrading, the arrival of the 130XE is something of a puzzle. The extra 64 Kbytes of RAM provide a lot more memory for the programmer and yet there are no programs available as yet to take advantage of it, even allowing for compatibility with the Atari software already on the market. Normally, one would expect the launch of such a machine to be a preliminary manoeuvre before a serious thrust at the small business market. However, the new Atari management has consistently denied any such intention. Perhaps the real reason for the launch of the 130XE is that Atari intended to pre-empt the launch of the Commodore 128, a Commodore 64 compatible machine that also has extra memory but is priced much higher.

ATARI 130XE

PRICE
£169.95 inc VAT

DIMENSIONS
350 x 233 x 63mm

CPU
6502C running at 1.79 MHz

MEMORY
128 Kbytes of RAM, 24 Kbytes of ROM

SCREEN
40 x 24 text display, 320 x 192 pixels (high-resolution) with 256 colours available

INTERFACES
Cartridge port, TV jack, composite monitor socket, two joystick ports, serial input/output port, expansion interface

LANGUAGES AVAILABLE
Atari BASIC, LOGO, FORTH, PILOT

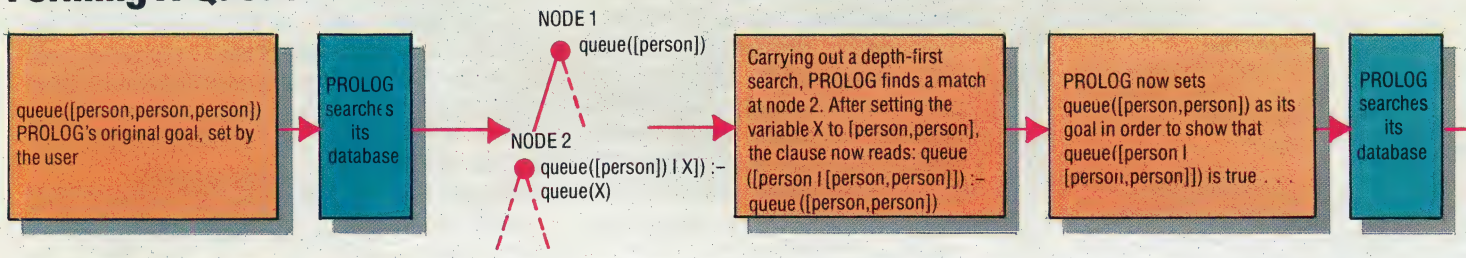
KEYBOARD
62 keys, including five pre-programmed function keys

DOCUMENTATION
The manual gives a full explanation of Atari BASIC, although the tone is still rather oversimplified. Explanation is given in the appendix of the interface configurations and how to 'bank' the other 64 Kbytes of RAM

STRENGTHS
The 130XE has the advantage of being able to trade on Atari's traditional strengths, yet possesses an additional 64 Kbytes of RAM that can be used as a 'silicon disk' for fast storage and retrieval

WEAKNESSES
The computer does not solve Atari's basic problem of lacking a large third-party software base. The fact that the 130XE is essentially a revamped version of a machine that has been around for some years may mean that it will not generate the interest in customers that the company needs

Forming A Queue


queue([person,person,person])
PROLOG's original goal, set by the user
PROLOG searches its database
NODE 1
queue([person])
NODE 2
queue([person]) I X]) :-
queue(X)
Carrying out a depth-first search, PROLOG finds a match at node 2. After setting the variable X to [person,person], the clause now reads: queue ([person I [person,person]]) :- queue ([person,person])
PROLOG now sets queue([person,person]) as its goal in order to show that queue([person I [person,person]]) is true . . .
PROLOG searches its database

Question Time
This flow-diagram shows PROLOG in action, answering a simple query posed by the user. Note that the variable X is given two different values during program execution, yet each value is retained. This is possible because PROLOG treats variables as local to each separate invocation of a clause

SEARCHING LOOKS

The flow of program control in PROLOG does not follow the familiar sequential pattern used in languages such as BASIC and PASCAL. We discuss PROLOG's tree-like search procedure, highlighting the general steps between defining a program's objective and achieving it.

While languages like BASIC and PASCAL have sequential flows of control, with control passing from statement to statement in strict top-to-bottom order (unless a loop or a GOTO interrupts it), the flow of control in PROLOG takes the form of a depth-first search through the program clauses.

To understand this more fully, think of the program as a tree, with the goal (proposition) to be proved at the root and all the sub-goals as choice-points where the lower branches divide. There are many ways to search a tree like this but the method used by PROLOG is to take the leftmost branch and follow it down as deeply as possible. As it tries each branch, it marks its trail, and when it reaches the bottom and can go no further, it backs up to the nearest choice-point, taking the leftmost branch that it has *not already taken*, and carries on down from there. In this way, the system will eventually explore every path through the tree and thus will have tried every possible way of proving the topmost goal.

An exhaustive depth-first search is guaranteed to cover all paths, but could be a lengthy business. In fact, PROLOG manages to save itself some work. A PROLOG clause is, as we saw in our previous instalment, a rule which says that the head goal is true if all of the subgoals are true:

```
goal:— subgoal1,subgoal2,subgoal3 . . .
   etc.
```

and so on. This is probably more easily understood if we write it as:

```
IF subgoal1 is true
   AND subgoal2 is true
   AND subgoal3 is true
   AND etc.
THEN goal is true.
```

Because the subgoals are ANDed together, the whole clause will fail if any one of them cannot be proved. So PROLOG works through the subgoals from left to right and, if it fails to prove one, will stop at that point and not bother with the rest.

Backtracking can make programs behave in a way that makes the order in which the code is written almost irrelevant. However, the advantage is that the flow of control, instead of being a major concern as it is in BASIC, has only a minor importance, leaving you to concentrate on the logical structure of your problem.

The emphasis that PROLOG places on a 'declarative' statement of the problem does not mean that you cannot see your programs as behaving procedurally. The PROLOG clause:

```
martian(X):— no. of limbs(X.7), no. of
   heads(X.2), can program in(X,cobol).
```

can be read declaratively as: 'X is a martian if X has seven limbs, two heads and can program in COBOL'. Its procedural reading would be: 'To prove that X is a martian, first prove that it has seven limbs, then prove it has two heads, then prove it can program in COBOL'.

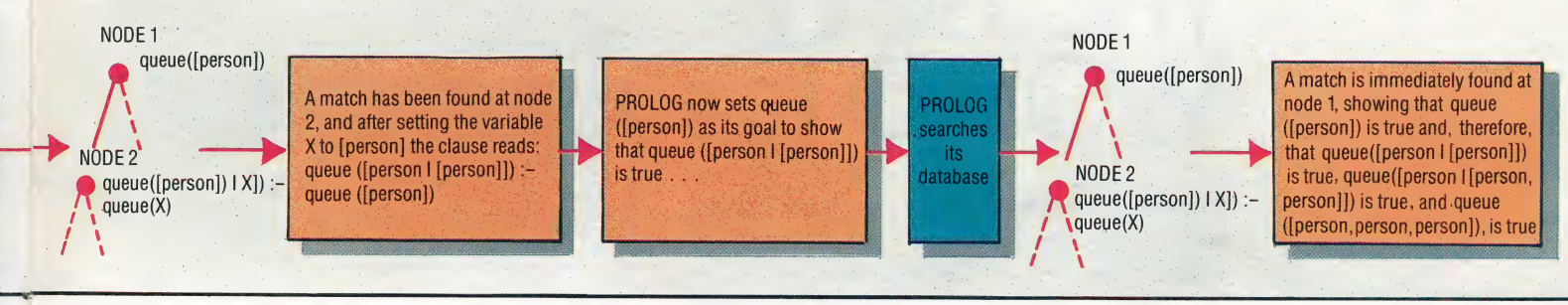
LISTS AND RECURSION

PROLOG is not as strongly 'typed' as most other languages. Nor is it fussy about the data types of arguments within its terms. So the term pred(Argument) could be used at different times with the variable Argument set to an integer, an 'atom' (such as marty, venusian, d24, and so on), or a list. However, PROLOG treats its data types in different ways, allowing, for instance, numbers to be arithmetically manipulated.

Anyone familiar with LOGO or LISP will have come across the list data type before and the special ways that are used for manipulating lists. A PROLOG list is written enclosed in square brackets with the list elements separated by commas. So [apple,pear,banana] is a list of fruits, [a,f,e,g,r,x] is a list of letters and so on.

To get inside lists, PROLOG lets us take them apart one element at a time by removing the first element. The notation [Head I Tail] describes a list with the element Head and the rest of the list in the list Tail. Applying 'I' to our list of fruits will give us [apple I [pear,banana]]. As you can see, lists can have other lists as members. As a special case, if we take the list [z80] with one element and break it up with I, we get [z80 I []]. The list in the tail is [], which represents the empty list.

PROLOG allows recursion, which, in fact, is the normal style for a PROLOG program. A recursive definition is one that defines something in terms of


NODE 1
queen([person])
NODE 2
queen([person]) I X]) :-
queen(X)
A match has been found at node 2, and after setting the variable X to [person] the clause reads: queue ([person I [person]]) :- queue ([person])
PROLOG now sets queue ([person]) as its goal to show that queue ([person I [person]]) is true . . .
PROLOG searches its database
NODE 1
queen([person])
NODE 2
queen([person]) I X]) :-
queen(X)
A match is immediately found at node 1, showing that queue ([person]) is true and, therefore, that queue([person I [person]]) is true, queue([person I [person, person]]) is true, and queue ([person, person, person]), is true

KEVIN JONES

itself. In PROLOG, we might write:

```
queue([person]).
queue([person I X]):— queue(X).
```

When we have two or more clauses, as we have here, which have the same head, they are known as a procedure. The queue procedure has a clause defining a queue as a list having one element, person. It then has a second clause informing us that a queue could also be a list with the element person at the head and with a list called X as its tail. We then see from the right-hand side of this clause that X must itself be a queue.

If we give PROLOG a goal such as:

```
queue([person,person,person]).
```

asking it to say whether the list [person,person,person] is a queue, it first looks for a clause in its database to match our goal. The first one it will find is queue([person]). This does not match because the lists are not identical, so it will scan down to the next clause: queue([person I X]):— queue(X). This does not match either and it fills in the values of the variables like this:

```
queue([person I [person,person]]):—
    queue([person,person]).
```

To show that the head goal is true, it must show that the subgoal is true. So PROLOG takes queue([person,person]) as its goal and begins scanning down the clauses *from the top* to find a match. Again, queue([person]) does not match, but the second clause does, giving:

```
queue([person I [person]]):—
    queue([person]).
```

An important point to note here is that the variable X, which on the first run through was set to [person,person], has now been set to [person]. Yet the first value of X has been retained.

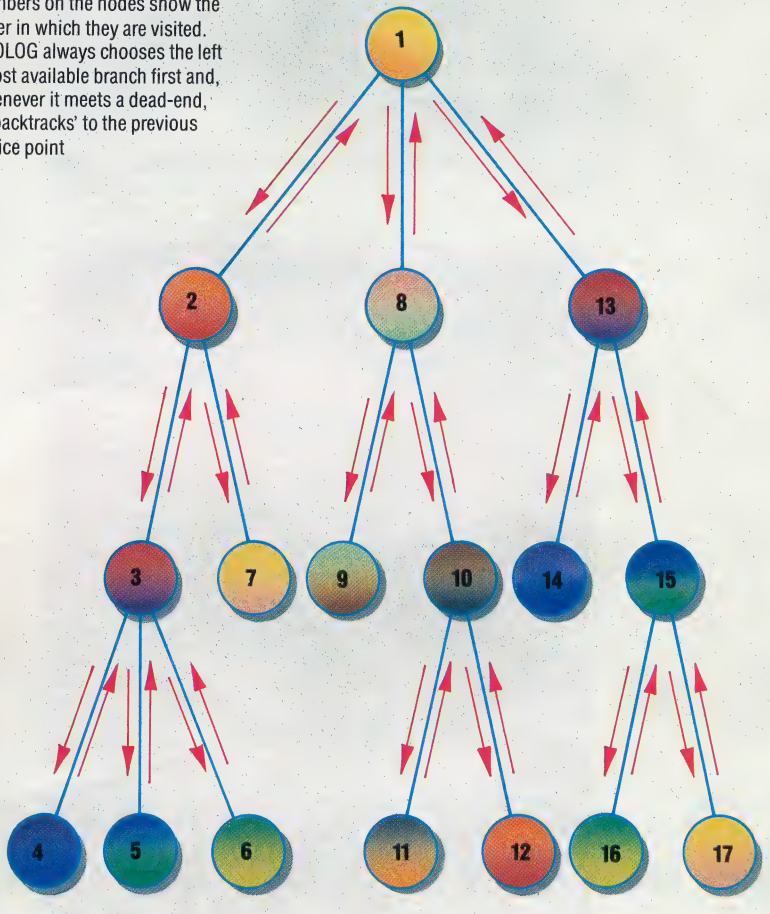
This is possible because variables are local to each separate invocation of a clause; thus every call to queue([person I X]) can be thought of as using a separate and unique variable. In PROLOG, there is no such thing as a global variable.

Having matched the head of this goal, to prove it we need to prove the subgoal on the right of the :— symbol, which is queue([person]). Another scan of the clause-base is begun and this time a match is made with queue([person]), which succeeds straight away and means that queue([person,person]) (our previous goal) is now shown to be true. This in turn means that queue([person, person,person]), the original goal, is true.

The queue procedure shows us a number of important things about PROLOG. For instance, the ordering of the clauses can be crucial. (Try placing the two clauses in the opposite order and see what happens when you try to prove a goal.) It also illustrates how giving extra clauses is the same as giving alternative ways of proving a goal, just as if we had used a logical OR between the clauses. This means that PROLOG does not need an OR operator — although most implementations provide one.

Moving Towards The Left

PROLOG uses a depth-first search to scan its database. The numbers on the nodes show the order in which they are visited. PROLOG always chooses the left-most available branch first and, whenever it meets a dead-end, it 'backtracks' to the previous choice point



KEVIN JONES

THE NEW WORLD I

The final instalments of our New World simulation game project are devoted to considering the BASIC flavours that will allow the game to run on the Spectrum, the BBC and the Amstrad micros. This article focuses on Spectrum flavours and includes the first section of the complete common listing.

The program was written on the Commodore 64 but uses minimal BASIC where possible. Problems in converting the program to run on the Spectrum lie in two main areas: first, the Spectrum will allow only single-letter variable names to be used with arrays or FOR . . . NEXT loop counters. We provide a conversion table here. Second, Spectrum string handling is unusual in that LEFT$, RIGHT$ and MID$ are not available, although each has a Spectrum equivalent. Spectrum owners should refer to the flavours given with each module for these conversions. In addition occurrences of PRINT CHR$(147) should be replaced by CLS and lines that wait for keypresses of the form:

<line no> GET I$:IF I$="" THEN <line no>

should be replaced by:

<line no> LET I$=INKEY$:IF I$="" THEN GO TO <line no>

In the next instalment we will give the second section of this full program listing.

Spectrum Variables Conversion Table

Microsoft Equivalent	Purpose	Spectrum
TS(,)	Crew type/strength	T(,)
WG()	Wage rates	W()
CC()	Crew type counts	Y()
PA()	Provisions purchased	A()
PC()	Provisions cost	C()
PN()	Provisioning needs	N()
OC()	Trading goods costs	O()
OA()	Trading goods amounts	G()
HR()	Half rations flags	H()
RR()	Voyage event flags	R()
AO()	Goods traded amounts	E()
EQ(,)	Exchange values	Q(,)
V1()	Values when leaving port	B()
V2()	Values on return	D()
S1	Reduce strengths counter	S
S3	Slow print counter	S
S4	Short delay counter	J
S5	Long delay counter	S

```
1 REM ***************
2 REM ** NEW WORLD **
3 REM **  TRADING  **
4 REM **   GAME    **
5 REM ***************
6 :
```

The first section of the program initialises variables and arrays that will be required in later sections

```
9 K$="  PRESS ANY KEY TO CONTINUE"
10 DIM TS(16,2): REM CREW TYPE/STRENGTH
11 CN=0: REM NO. OF CREW
12 MO=2000: REM START MONEY
13 DIMWG(5):WG(1)=10:WG(2)=25:WG(3)=15:WG(4)=20:WG
(5)=15: REM WAGES
14 WT=0: REM WEEKLY WAGE BILL
15 CM=16:REM MAX CREW
16 DIMC$(5):C$(1)="SAILOR":C$(2)="DOCTOR":C$(3)="M
ECHANIC"
17 C$(4)="NAVIGATOR":C$(5)="COOK"
18 DIMCC(5): REM COUNT OF EACH CREW TYPE
20 DIMU$(4):U$(1)="KILO":U$(2)="KILO":U$(3)="KILO"
:U$(4)="BARREL"
21 DIMP$(4):P$(1)="VEG":P$(2)="FRUIT":P$(3)="MEAT"
:P$(4)="WATER"
22 DIMPA(4)
23 DIMPC(4):PC(1)=.5:PC(2)=1:PC(3)=2:PC(4)=.5
24 DIMPN(4):PN(1)=2:PN(2)=1:PN(3)=1:PN(4)=.5
30 DIMOA(6)
31 DIMO$(6)
32 O$(1)="BOTTLE OF MEDICINE":O$(2)="GUN":O$(3)="B
AG OF SALT"
33 O$(4)="BALE OF CLOTH":O$(5)="KNIFE":O$(6)="JEWEL"
34 DIMOC(6)
35 OC(1)=1:OC(2)=5:OC(3)=.2
36 OC(4)=2:OC(5)=.5:OC(6)=1
40 JL=8:REM LENGTH OF VOYAGE
41 EW=0: REM EXTRA WEEKS
42 DIMRR(16)
43 REM INDICATORS TO SHOW IF RANDOM EVENT(N) ALREA
DY OCCURED
44 RC=0
45 REM COUNT OF RANDOM EVENTS SO FAR
46 RM=13
47 G$="N":REM GOOD WEATHER INDICATOR FOR USE IN MU
TINY FACTOR
48 A$="N":B$="N"
49 DIMM(6):REM INDS TO SHOW IF MAJOR EVENTS HAVE B
EEN DONE
60 DIMT$(3):T$(1)="PEARLS":T$(2)="CARVINGS":T$(3)=
"SPICES"
61 DIMV1(3):V1(1)=2:V1(2)=2:V1(3)=1
62 DIMV2(3):V2(1)=2+(INT(RND(1)*1)/2):V2(2)=2+(INT
(RND(1)*3)-1)
63 V2(3)=2+(INT(RND(1)*1)/2)
64 DIMEQ(4,3)
65 EQ(1,1)=0.5:EQ(1,2)=0.5:EQ(1,3)=1
66 EQ(2,1)=5:EQ(2,2)=5:EQ(2,3)=10
67 EQ(3,1)=3:EQ(3,2)=3:EQ(3,3)=6
68 EQ(4,1)=2:EQ(4,2)=2:EQ(4,3)=4
69 DIMAO(3)
80 PRINTCHR$(147):S$="  NEW WORLD TRADING GAME*":G
OSUB9100:PRINT
81 GOSUB9200
82 S$="YOU ARE THE CAPTAIN OF A SHIP*" :GOSUB9100:
PRINT
83 S$="SAILING TO THE NEW WORLD. THE*":GOSUB9100:P
RINT
84 S$="JOURNEY IS SAID TO TAKE EIGHT*":GOSUB9100:P
RINT
85 S$="WEEKS,BUT MAY TAKE LONGER. YOU*":GOSUB9100:
PRINT
86 S$="MUST HIRE A CREW,PAY THEM, BUY*":GOSUB9100:
PRINT
87 S$="PROVISIONS,EQUIPMENT AND GOODS*":GOSUB9100:
PRINT
88 S$="FOR TRADING. YOU HAVE 2000 GOLD*":GOSUB9100
:PRINT
89 S$="PIECES TO SPEND.*":GOSUB9100:PRINT:GOSUB9200

90 PRINT:S$="         GOOD LUCK!*":GOSUB9100:GOSUB
9200:PRINT
91 S$=K$:GOSUB9100
94 GETI$:IFI$=""THEN94
95 GOSUB9200
```

This section of the main program calls the subroutines that allow the player to hire a crew, stock the ship with provisions for the voyage and buy the goods to be traded in the New World

```
500 GOSUB1000
550 GOSUB2000
600 GOSUB3000
610 PRINTCHR$(147)
615 S$="YOU ARE NOW READY TO START*":GOSUB9100
625 S$="THE VOYAGE.*":GOSUB9100
630 GOSUB9200
635 PRINT:S$="YOU HAVE THE FOLLOWING CREW:*":GOSUB
9100
```

```
640 GOSUB9200
645 FORT=1TO5
650 IFCC(T)=0THEN670
655 PRINTCC(T);
660 PRINTC$(T);
662 IF CC(T)=1THENPRINT" ":GOTO668
664 PRINT"S"
668 GOSUB9200
670 NEXT
674 GOSUB9200
675 PRINT:S$="AND THE FOLLOWING PROVISIONS:*":GOSU
B9100
680 GOSUB9200
685 FORT=1TO4
690 IFPA(T)=0THEN710
695 PRINTPA(T);U$(T);"S OF ";
700 PRINTP$(T)
708 GOSUB9200
710 NEXT
715 GOSUB9200
720 PRINT:S$="YOU HAVE ALSO GOT:*":GOSUB9100
725 GOSUB9200
730 IFOA(1)=0THEN740
733 IFOA(1)=1THENS$="BOTTLE OF MEDICINE *":GOTO735
734 S$="BOTTLES OF MEDICINE*"
735 PRINTOA(1);:GOSUB9100
736 GOSUB9200
740 IFOA(2)=0THEN750
743 IFOA(2)=1THENS$="GUN*":GOTO745
744 S$="GUNS*"
745 PRINTOA(2);:GOSUB9100
746 GOSUB9200
750 IFOA(3)=0THEN760
753 IFOA(3)=1THENS$="BAG OF SALT*":GOTO755
754 S$="BAGS OF SALT*"
755 PRINTOA(3);:GOSUB9100
756 GOSUB9200
760 IFOA(4)=0THEN770
763 IFOA(4)=1THENS$="BALE OF CLOTH*":GOTO765
764 S$="BALES OF CLOTH*"
765 PRINTOA(4);:GOSUB9100
766 GOSUB9200
770 IFOA(5)=0THEN780
773 IFOA(5)=1THENS$="KNIFE*":GOTO775
774 S$="KNIVES*"
775 PRINTOA(5);:GOSUB9100
776 GOSUB9200
780 IFOA(6)=0THEN790
783 IFOA(6)=1THENS$="JEWEL*":GOTO785
784 S$="JEWELS*"
785 PRINTOA(6);:GOSUB9100
786 GOSUB9200
790 GOSUB9200
792 PRINT:PRINT"YOU HAVE ";MO;" GOLD PIECES LEFT"
796 GOSUB9200
797 S$="PRESS ANY KEY TO START VOYAGE*"
798 GOSUB9100
799 GETI$:IF I$=""THEN799
800 WT=0 :REM ZEROISE WAGE TOTAL
801 H$="N" :REM HALF RATION INDICATOR
802 DIMHR(4):HR(1)=1:HR(2)=1:HR(3)=1:HR(4)=1
```

The main voyage loop starts here, using the variable WK to count off the number of weeks taken

```
820 WK=1
825 GOSUB4000: REM CREW STATUS REPORT
830 GOSUB4200:REM PROVISIONS REPORT
835 GOSUB4300:REM OTHER GOODS REPORT
840 GOSUB9200:PRINTCHR$(147)
842 PRINT:PRINT:PRINT
843 S$="IT IS ESTIMATED THAT THE VOYAGE*":GOSUB9100
844 PRINT"WILL TAKE A FURTHER";INT(JL-WK+1);"WEEKS"
845 GOSUB9200
846 PRINT:S$=K$:GOSUB9100
847 GETI$:IFI$=""THEN847
850 GOSUB5000:REM CHECK WAGE BILL
855 GOSUB5100: REM ISSUE RATIONS
860 GOSUB5500
861 REM GO TO GENERATE RANDOM EVENTS
870 GOSUB6500:REM GOTO MAJOR CONTINGENCY
875 IFHR(3)=.5ANDRND(1)<.5THENPRINTCHR$(147):GOSUB
6050
878 REM ALBATROSS IF SHORT OF MEAT
879 GOSUB7200
880 GOSUB 5300: REM END-OF-WEEK REPORT
889 WK=WK+1:IFWK<=JLTHEN825
890 REM ARRIVAL AT NEWWORLD
891 GOSUB10000
892 GOSUB10070
893 GOSUB10300
894 GOSUB10500
999 END
```

The main program ends here. The rest of the program is in the form of subroutines that are called from this main section

```
1000 PRINTCHR$(147):PRINT" STAGE 1 - HIRING CREW"
1010 PRINT" ----------------------"
1012 PRINT
1015 GOSUB9200
1020 PRINT:PRINT"CREW TYPES AVAILABLE:"
1025 GOSUB9200
1030 PRINT
1040 PRINT"TYPE DESCRIPTION WAGES PER WEEK"
1050 PRINT"---- ----------- ---------------"
1060 PRINT"  1   SAILOR       10 GOLD PCS"
1070 PRINT"  2   DOCTOR       25 GOLD PCS"
1080 PRINT"  3   MECHANIC     15 GOLD PCS"
1090 PRINT"  4   NAVIGATOR    20 GOLD PCS"
1100 PRINT"  5   COOK         15 GOLD PCS"
1105 GOSUB9200
1110 PRINT:PRINT:PRINT
1120 S$="ENTER CREW TYPE REQUIRED(1-5)*":GOSUB9100
1122 S$="OR 'F' TO FINISH HIRING*":GOSUB9100:PRINT
:INPUTI$
1125 CT=VAL(I$)
1128 IFLEFT$(I$,1)="F"THENPRINT:PRINT"END OF CREW
HIRE.":GOSUB9200:GOTO1310
1130 IFCT>0ANDCT<6THEN1150
1139 PRINT:PRINT
1140 PRINTI$;:S$=" IS NOT A CREW TYPE*":GOSUB9100
1142 GOSUB9200
1145 S$="PLEASE ENTER AGAIN"
1146 GOSUB9100
1147 GOTO1300
1150 PRINT:PRINT
1155 CN=CN+1:REM CREW HIRED SO FAR
1156 TS(CN,1)=AT:REM CREW TYPE
1157 TS(CN,2)=100:REM STARTING STRENGTH
1158 WT=WT+WG(CT):REM TOTAL WAGES
1159 CC(CT)=CC(CT)+1:REM CREW TYPE COUNT
1160 S$="CREW SO FAR:"
1170 FORT=1TO5
1180 PRINTS$;CC(T);" ";C$(T);
1185 IFCC(T)>1ORCC(T)=0THENPRINT"S":GOTO1189
1186 PRINT" "
1189 S$="  "
1190 NEXT
1195 PRINT: PRINT"TOTAL WEEKLY WAGE BILL ";WT
1200 IFCN=CM-1THENPRINT:S$="ONLY ONE MORE CREW*":G
OSUB9100:GOTO1295
1202 IFCN=CMTHENPRINT:S$="    SHIP NOW FULL!!":GOS
UB9100:GOTO1310
1295 REM
1300 GOTO1015
1310 PRINT: S$=K$:GOSUB9100:PRINT:   GOSUB9200
1320 GETI$:IFI$=""THEN1320
1999 RETURN
2000 PRINTCHR$(147)
2010 S$="   STAGE 2 - PROVISIONING*"
2015 GOSUB9100
2020 S$="  -------------*"
2025 GOSUB9100
2030 GOSUB9200:PRINT
2040 PRINT"YOU'VE HIRED A CREW OF ";CN;"."
2045 GOSUB9200:GOSUB9200
2050 FORT=1TO4
2055 PRINT
2060 PRINT"EACH CREW MEMBER WILL NEED "
2070 PRINT"AT LEAST ";PN(T);" ";U$(T);
2075 IFPN(T)=1THENPRINT" ";:GOTO2085
2080 PRINT"S";
2085 PRINT" OF ";P$(T)
2086 PRINT"AT ";PC(T);" GOLD PCS PER ";U$(T)
2090 PRINT"FOR EACH WEEK OF THE JOURNEY."
2095 GOSUB9200:PRINT:GOSUB9200
2100 PRINT"HOW MANY ";U$(T);"S OF ";P$(T)
2110 S$="DO YOU WANT TO BUY*":GOSUB9100
2120 PRINT
2130 INPUTI$
2140 PA(T)=VAL(I$):GOSUB9200
2150 IFPA(T)>((CN*8*PN(T))-1) THEN2260
2160 IFPA(T)=0THENPRINT"IF YOU DON'T BUY ANY":GOTO
2180
2170 PRINT"IF YOU ONLY BUY ";PA(T);U$(T);
2175 IFPA(T)=1THENPRINT" OF":GOTO2180
2176 PRINT"S OF"
2180 PRINTP$(T);",":GOSUB9200
2190 PRINT"SOMEONE MIGHT GET ";
2200 S$="HUNGRY"
2210 IFT=4THENS$="THIRSTY"
2220 PRINTS$;"!":GOSUB9200
2230 S$= "DO YOU WANT TO TRY AGAIN":GOSUB9100
2240 INPUTP$:P$=LEFT$(P$,1)
2242 IFP$<>"Y"ANDP$<>"N"THEN2230
2245 IFP$="N"THEN2400
2250 PA(T)=0:T=T-1:GOTO2410
2260 IFPA(T)*PC(T)>MOTHEN2270
2265 GOTO2400
2270 S$="YOU DON'T HAVE ENOUGH MONEY FOR":GOSUB910 0
2280 PRINTPA(T)
2290 PRINTU$(T);"S OF ";P$(T):GOSUB9200
2300 S$="PLEASE TRY AGAIN*":GOSUB9100:PA(T)=0:T=T-
1:GOTO2410
2400 MO=MO-(PA(T)*PC(T))
2410 PRINT:S$="PROVISIONS SO FAR:*":GOSUB9100
2412 GOSUB9200
2415 FORTT=1TO4
2420 PRINTPA(TT);U$(TT);
2430 IFPA(TT)=1THENPRINT"  OF ";:GOTO2440
2435 PRINT"S OF ";
2440 PRINTP$(TT)
2450 GOSUB9200
```

```
2460 NEXT
2480 PRINT"MONEY LEFT = ";MO;" GOLD PIECES"
2485 GOSUB9200:GOSUB9200
2490 NEXTT
2500 GOSUB9201:PRINT:S$="END OF PROVISIONING*":GOS
UB9100:GOSUB9200
2510 PRINT: S$=K$:GOSUB9100:PRINT:   GOSUB9200
2520 GETI$:IF I$=""THEN2520
2999 RETURN
3000 PRINTCHR$(147): REM STAGE 3
3002 GOSUB9200
3005 PRINT"     STAGE 3 - OTHER GOODS"
3010 PRINT"     -------   -----------"
3020 GOSUB9200
3025 PRINT
3030 S$="THERE ARE OTHER THINGS THAT MAY*":GOSUB9100
3035 S$="BE USEFUL ON THE VOYAGE. FOR  *":GOSUB9100
3040 S$="EXAMPLE MEDICINE AND TRADING  *":GOSUB9100
3046 GOSUB9200
3050 S$="YOU MAY ALSO NEED WEAPONS.*":GOSUB9100
3055 GOSUB9200:GOSUB9200
3060 FORT=1TO6
3065 PRINT
3070 PRINT"A ";O$(T);
3075 S$=" COSTS*":GOSUB9100
3080 PRINTOC(T);
3081 PRINT" GOLD PIECE";
3085 IFOC(T)=1THENPRINT" ":GOTO3090
3086 PRINT"S"
3090 GOSUB9200
3095 S$="WOULD YOU LIKE TO BUY (Y OR N)*":GOSUB9100
3110 INPUTP$:P$=LEFT$(P$,1)
3115 IFP$<>"Y"ANDP$<>"N"THEN3095
3120 IFP$="N"THEN3175
3125 GOSUB9200
3130 S$="HOW MANY DO YOU WANT*":GOSUB9100
3135 INPUTI$
3140 TT=VAL(I$)
3145 IFOC(T)*TT>MOTHEN3150
3147 GOTO3160
3150 S$="YOU DON'T HAVE ENOUGH MONEY*":GOSUB9100
3152 GOSUB9200
3154 S$="PLEASE ENTER AGAIN  *":GOSUB9100
3155 GOSUB9200:GOTO3130
3160 MO=MO-(OC(T)*TT)
3165 OA(T)=TT
3170 GOSUB9200
3175 PRINT
3176 PRINT"MONEY LEFT = ";MO
3200 GOSUB9200:NEXT T
3205 GOSUB9200:PRINT:PRINT
3210 S$="END OF STAGE 3*":GOSUB9100
3220 GOSUB9200:PRINT
3230 S$=K$:GOSUB9100
3240 GETI$:IFI$=""THEN3240
3999 RETURN
```

The following subroutines are called from within the main voyage loop to analyse the current status of the ship and crew and make a weekly report to the player

```
4000 REM CREW STATUS REPORT
4010 PRINTCHR$(147)
4020 S$="        CAPTAINS LOG*":GOSUB9100
4025 S$="        ------------*":GOSUB9100
4030 GOSUB9200
4035 PRINT"AT THE START OF WEEK";WK
4040 S$="THE STATE OF THE CREW IS:*":GOSUB9100
4045 GOSUB9200:PRINT
4055 PRINT
4060 FORT=1TO16
4070 IFTS(T,1)=0THEN4110
4075 PRINTC$(TS(T,1));"    (";
4078 IFTS(T,2)=-999THENS$="DEAD !!!!!!!)*":GOTO4099
4080 IFTS(T,2)>75THENS$="VERY HEALTHY)*":GOTO4099
4085 IFTS(T,2)>50THENS$="HEALTHY)*":GOTO4099
4095 IFTS(T,2)>25THENS$="SICK ! )*":GOTO4099
4098 S$="VERY SICK !! )*"
4099 GOSUB9100:GOSUB9200
4110 NEXT
4115 GOSUB9200:PRINT
4119 WW=0
4120 FORT=1TO5
4130 WW=WW+(CC(T)*WG(T))
4135 NEXT
4140 S$="WAGE BILL FOR THE WEEK*":GOSUB9100
4145 PRINTWW;"GOLD PIECES"
4150 GOSUB9200
4155 WT=WT+WW
4160 S$="TOTAL WAGES FOR VOYAGE SO FAR:*":GOSUB9100
4165 PRINTWT;"GOLD PIECES"
4170 GOSUB9200
4175 PRINT"MONEY LEFT = ";MO;"GOLD PIECES"
4180 PRINT:S$=K$:GOSUB9100
4190 GETI$:IFI$=""THEN4190
4199 RETURN
4200 REM PROVISIONS REPORT
4205 PRINTCHR$(147)
4206 PRINT"AT THE START OF WEEK";WK:GOSUB9200
4210 S$="YOU HAVE THE FOLLOWING*":GOSUB9100
4215 S$="PROVISIONS LEFT:*":GOSUB9100
4220 PRINT:GOSUB9200
4225 FORT=1TO4
4226 IFPA(T)=0ORPA(T)=-999THEN4240
4230 PRINTPA(T);U$(T);"S OF ";P$(T)
4232 X=PA(T)/(CN*PN(T))
4235 PRINT"(ENOUGH FOR";INT(X);" WEEKS)"
4239 GOSUB9200
4240 NEXT
4290 PRINT:S$=K$:GOSUB9100
4295 GETI$:IFI$=""THEN4295
4299 RETURN
4300 REM OTHER GOODS REPORT
4305 PRINTCHR$(147)
4306 PRINT"AT THE START OF WEEK";WK:GOSUB9200
4310 S$="YOU ALSO HAVE:*":GOSUB9100
4320 PRINT:GOSUB9200
4322 IFOA(1)=0THEN4332
4325 PRINTOA(1);:S$="BOTTLES OF MEDICINE*":GOSUB9100
4330 GOSUB9200
4332 IFOA(2)=0THEN4342
4335 PRINTOA(2);:S$="GUNS*":GOSUB9100
4340 GOSUB9200
4342 IFOA(3)=0THEN4352
4345 PRINTOA(3);:S$="BAGS OF SALT*":GOSUB9100
4350 GOSUB9200
4352 IFOA(4)=0THEN4362
4355 PRINTOA(4);:S$="BALES OF CLOTH*":GOSUB9100
4360 GOSUB9200
4362 IFOA(5)=0THEN4372
4365 PRINTOA(5);:S$="KNIVES*":GOSUB9100
4370 GOSUB9200
4372 IFOA(6)=0THEN4380
4375 PRINTOA(6);:S$="JEWELS*":GOSUB9100
4380 GOSUB9200:PRINT
4382 PRINT"YOU HAVE";MO;:S$=" GOLD PIECES LEFT*":G
OSUB9100
4384 GOSUB9200
4397 PRINT:S$=K$:GOSUB9100
4398 GETI$:IFI$=""THEN4398
4399 RETURN
5000 REM CHECK WAGE BILL
5005 IFWT>MOTHEN5010
5008 GOTO5099
5010 PRINTCHR$(147)
5020 PRINT:PRINT:PRINT
5025 S$="THE CREW HAVE HEARD A RUMOUR*":GOSUB9100
5030 S$="THAT YOU DON'T HAVE ENOUGH*":GOSUB9100
5035 S$="GOLD TO PAY THEM AT THE END*":GOSUB9100
5040 S$="OF THE VOYAGE.*":GOSUB9100
5045 GOSUB9200:PRINT
5050 S$="THEY ARE GETTING ANGRY !!*":GOSUB9100
5055 GOSUB9200:PRINT
5060 S$="LET'S HOPE YOU MANAGE TO MAKE*":GOSUB9100
5065 S$="A TRADING PROFIT!*":GOSUB9100
5066 GOSUB9200
5070 PRINT:S$=K$:GOSUB9100
5080 GETI$:IFI$=""THEN5080
5099 RETURN
5100 REM ISSUE RATIONS
5103 PRINTCHR$(147)
5105 S$="       ISSUING RATIONS*":GOSUB9100
5106 S$="       ---------------*":GOSUB9100
5107 GOSUB9200:PRINT"WEEK";WK:PRINT
5108 H$="N"
5110 FORT=1TO4
5112 HR(T)=1
5115 IFPA(T)>0THEN5180
5120 PRINT"NO ";P$(T);" LEFT!!!":GOSUB9200
5130 S$="THE CREW IS GETTING WEAKER !!*":GOSUB9100
5135 WF=10:GOSUB9300
5139 GOTO5290
5180 X=(PN(T)*CN)*(JL-WK+1)
5185 IFPA(T)<XTHEN5200
5190 GOTO5270
5200 PRINT"RUNNING SHORT OF ";P$(T)
5205 GOSUB9200
5210 S$="DO YOU WANT TO PUT THE CREW ON*":GOSUB910 0
5215 PRINT"HALF RATIONS OF ";P$(T)
5220 INPUTI$:I$=LEFT$(I$,1)
5221 IFI$<>"Y"ANDI$<>"N"THEN5220
5225 IFI$="N"THEN5270
5230 HR(T)=.5:H$="Y"
5240 WF=5:GOSUB9300
5250 S$="THE CREW IS GETTING WEAKER!*":GOSUB9100
5270 X=PN(T)*HR(T)*CN
5272 IFX>PA(T)THENX=PA(T)
5275 PA(T)=PA(T)-X
5280 IFPA(T)=0THENPA(T)=-999
5285 PRINTX;U$(T);"S OF ";P$(T);" ISSUED"
5290 PRINT:GOSUB9200:NEXT
5295 PRINT:S$=K$:GOSUB9100
5298 GETI$:IFI$=""THEN5298
5299 RETURN
```

IAN McKINNELL

CHANNELS OF THOUGHT

We continue our series on the Spectrum's operating system with a look at how the channels through which the micro sends data to the screen and ZX printer are selected and controlled. Our discussion takes in the use of system variables in handling different aspects of the display.

On the Sinclair Spectrum the usual means of input is the keyboard and the usual outputs are either a television screen or a ZX printer. In the Spectrum system, each of these hardware items is called an input or output *channel*. The screen, for example, is referred to as an output channel. The actual data flowing to or from the computer, in the form of characters to the screen and information from the keyboard, is called a *stream*. A data stream is capable of being directed into different hardware channels, assuming that the channel hardware is capable of handling the stream in the correct fashion.

We'll be taking a closer look at streams and channels in our next instalment. In the meantime, we'll begin our examination of the Spectrum I/O system with a look at the 'output a character' routine, which can be found at address &0010. The character sent to this routine is forwarded to the screen or printer, depending upon which channel has been previously selected. Our diagram shows the various channels on an unexpanded Spectrum and the stream numbers that are associated with them. Channels are named with a single letter, and streams are described by a number. In the unexpanded Spectrum only streams 0 to 3 are active. The Spectrum OS allocates streams to channels as shown in the table in the margin.

Stream	Channel
0	K
1	K
2	S
3	P

Thus, when we want to output a character to a particular device we must first tell the Spectrum OS which stream we want it to be in. To write to the screen, we would select stream 2, as this is the one associated with channel S, and to do this we use a ROM routine at address &1601 to tell the OS which stream we want to select. The stream number is placed in the A register, before calling the routine, which then opens up the hardware channel currently associated with that stream number. For example, to open channel S for output, we execute the following instructions:

```
LD      A,2
CALL    &1601
```

Once a channel is open, sending a character to it is simply a matter of putting the character's code into the A register and then executing an RST instruction to call the routine at address &0010.

This is similar in some respects to the OSWRCH call on the BBC Micro, with the exception that we call an address in the ROM directly rather than via a vector. Channel S operates over the area of the

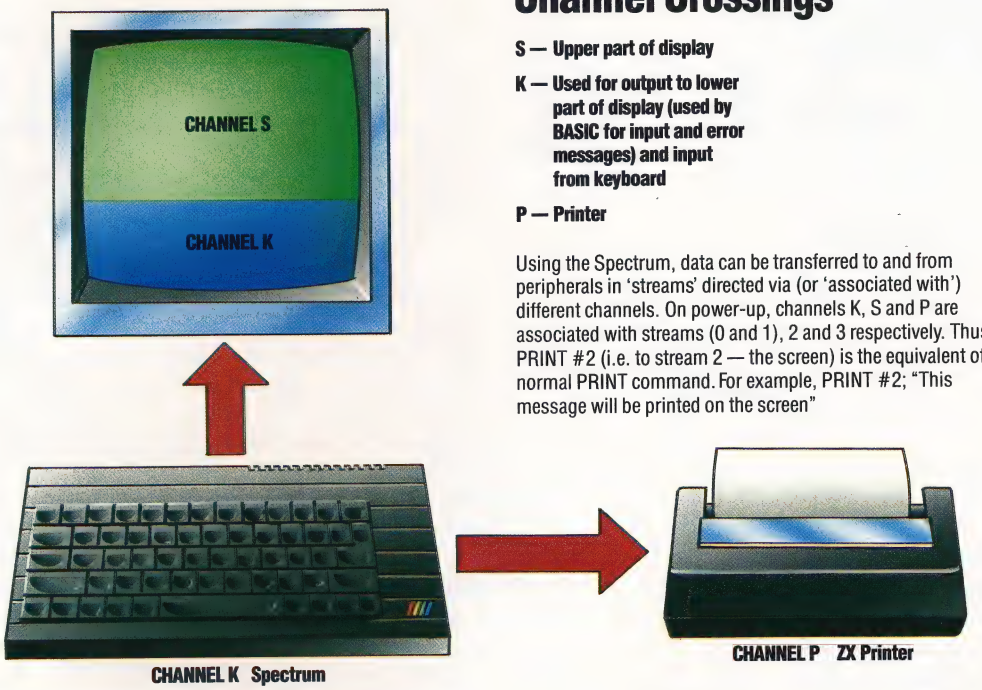
Channel Crossings

S — Upper part of display

K — Used for output to lower part of display (used by BASIC for input and error messages) and input from keyboard

P — Printer

Using the Spectrum, data can be transferred to and from peripherals in 'streams' directed via (or 'associated with') different channels. On power-up, channels K, S and P are associated with streams (0 and 1), 2 and 3 respectively. Thus, PRINT #2 (i.e. to stream 2 — the screen) is the equivalent of the normal PRINT command. For example, PRINT #2; "This message will be printed on the screen"



KEVIN JONES

screen that is accessible via the normal BASIC PRINT statements. However, it is also possible to access the lower two lines of the screen that are normally used by the BASIC interpreter for the output of error messages and prompts. As can be seen from the diagram, these lines form part of channel K.

To send a character to these lines, we simply open channel K using:

```
LD      A,0
CALL    &1601
```

Similarly, the printer channel can be opened by loading the A register with 3.

As well as standard characters, we can send control characters to any channel. Clearly, the effect that these control characters have will depend upon the channel in use, but it does mean that we can do the machine code equivalents of PRINT AT, PRINT INK, PRINT PAPER and so on. The following table shows some useful control codes and their functions when passed to channel S or K. Obviously, some will have no effect if sent to the ZX Printer (via channel P).

Code	Parameters	Effect
8	—	Cursor left one space
9	—	Cursor right one space
10	—	Cursor down one line
11	—	Cursor up one line
12	—	Delete
13	—	ENTER
16	n	INK n (needs one more byte)
17	n	PAPER n (needs one more byte)
18	n	FLASH n (needs one more byte)
19	n	BRIGHT n (needs one more byte)
20	n	INVERSE n (needs one more byte)
21	n	OVER n (needs one more byte)
22	nn	AT y,x (needs two more bytes for the x and y co-ordinates)
23	n	TAB n (needs one more parameter)

The extra bytes requested by some control codes are the parameters that would normally follow them in a BASIC statement. For example, to execute a PAPER 3 command, we'd simply send the bytes 17 and 3 to channel S. The following segment of a program shows the machine code equivalent of PRINT AT 10,10;"A".

```
3E02      10      ld   a,2              ;open channel S by
CD0116    20      call #1601            ;selecting stream 2
3E16      30      ld   a,22             ;token for AT
D7        40      rst  #10              ;output a char routine
3E0A      50      ld   a,10             ;y co-ordinate
D7        60      rst  #10
3E0A      70      ld   a,10             ;x co-ordinate
D7        80      rst  #10
3E41      90      ld   a,65             ;ASCII code for 'A'
D7       100      rst  #10
C9       110      ret
```

Not too difficult, is it? One notable exception in the list of control codes is the absence of a code for CLS. To clear the screen, we must call another ROM routine at address &06DB. It is essential that channel S is open before we call it, and it's also necessary to re-open channel S after we've used it if we want to print anything else to the screen. This routine will clear the screen:

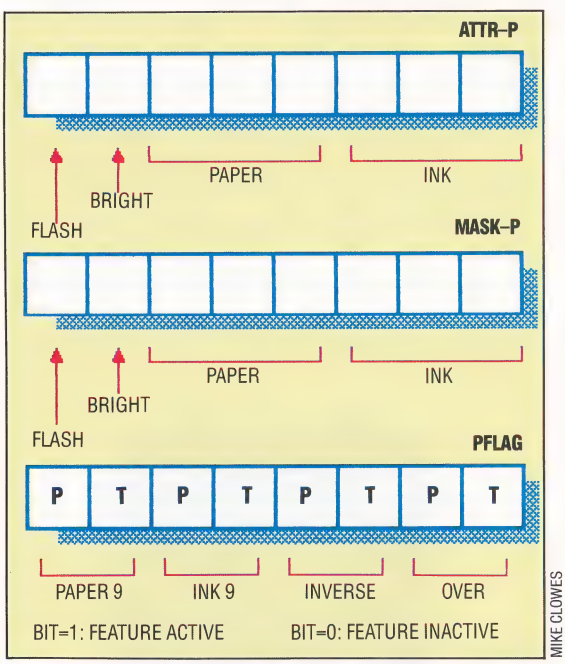
```
3E02      10      ld   a,2
CD0116    20      call #1601            ;open channel S
CDDB06    30      call #06db            ;clear the screen
3E02      40      ld   a,2
CD0116    50      call #1601            ;re-open channel S
C9        60      ret
```

A useful attribute of the 'output a character' routine on a standard Spectrum is that numbers passed to it that represent BASIC keywords are expanded by the routine and printed in full. Thus:

```
LD      A,249
RST     &10
```

will — assuming channel S, P or K has been selected — print the keyword RANDOMISE to the screen.

Graphic commands, such as PAPER, INK and BRIGHT, issued via the 'output a character' routine are operative only over that sequence of output characters — they are said to be *temporary colour items*. The PAPER n command, if issued outside of a PRINT statement, is a *permanent* command, and it remains operative until another PAPER command is issued.



The two system variables of interest are called ATTR-P and MASK-P. The diagram above shows how these system variables control different aspects of the display. In ATTR-P, the three bits that control PAPER and INK colour on a permanent basis are given the values shown at the top of the next column. If the bit for FLASH or BRIGHT is set to one, then that feature is operational. ATTR-P is at address 23693.

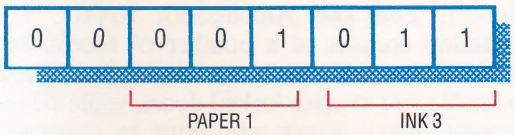
MASK-P is at address 23694 and any bit set to one in this byte will ensure that attributes on the screen at the relevant print position are not altered by the contents of ATTR-P. A further useful system variable is called PFLAG, and this is found at address 23697. This variable is also shown in the

Bits	Colour
000	Black
001	Blue
010	Red
011	Magenta
100	Green
101	Cyan
110	Yellow
111	White

diagram. It is used by the OS to indicate PAPER 9, INK 9, INVERSE and OVER.

There are two other system variables — ATTR-T and MASK-T — at addresses 23695 and 23696, respectively. These are arranged in a similar way to ATTR-P and MASK-P, but control the temporary colours (those in use in PRINT statements or those set up by sending control codes through the RTS &10 routine).

Using ATTR-P and MASK-P to set up colours is quite easy; we simply manipulate the contents of the system variables, altering only those bits that we need to. This is easily done using the Z80's AND and OR logical instructions. So, to execute a permanent PAPER 1:INK 3 command, we would set ATTR-P up to:



using this piece of code:

```
LD      A,11
LD      (23693),A
RET
```

Of course, as with BASIC permanent PAPER and INK commands, the new colours will not affect what's been printed before and the screen will not go to the new PAPER colour until a CLS command, or its equivalent, has been executed.

The graphics routines on the Spectrum are part of the BASIC interpreter program. We will now take a look at the PLOT and DRAW operations. When using these routines it is easy to bring about colour changes by altering the ATTR-P and MASK-P variables as we have already described here.

The routines themselves are easy to use. The first of these, PLOT, is called at address &22E5. Its co-ordinates are passed to it in the BC register pair, (B holding the y co-ordinate and C the x co-ordinate). Thus, to execute the PLOT 100,100 command from a machine code program we'd simply execute these instructions:

```
LD      B,100    ;y co-ordinate
LD      C,100    ;x co-ordinate
CALL    &22E5    ;do it
RET
```

Colour changes are easily performed; the following routine plots a red dot on the screen, and then restores ATTR-P to its previous state before returning to BASIC.

```
3A8D5C    10        ld   a,(23693)      ;get ATTR-P in A reg
F5        20        push af             ;and save on stack
E6F8      30        and  248            ;clear low 3 bits to 0
F602      40        or   2              ;set bit 1 to give red ink
0664      50        ld   b,100          ;y co-ordinate
0E64      60        ld   c,100          ;x co-ordinate
CDE522    70        call #22e5          ;call PLOT
F1        80        pop  af             ;restore original
328D5C    90        ld   (23693),a      ;ATTR-P contents
C9        100       ret
```

So far we've concentrated on channel S. What about channel K, which also has output facilities? We can write to the lower part of the screen if we want to, but what we put there will be overwritten when the operating system or interpreter generates a message. For input using this channel, we don't even need to call a ROM routine! The keyboard is scanned once every 20 microseconds and some system variables are affected, depending on whether a key is being pressed or not. Two of particular use are LAST-K (at address 23560), which holds the character code of the last key pressed, and a system variable at 23556, which holds the value 255 if a key is *not* being pressed at that point in time. The routine here can thus be used to wait until a key is pressed and then return the character code in the A register. It simply checks the contents of address 23556 until it doesn't equal 255. The value in LAST-K will then be that of the key being pressed.

```
          10 ; GET-CHAR routine
3A045C    20 key:   ld   a,(23556)      ;check to see if
FEFF      30        cp   255            ;key is being pressed
28F9      40        jr   z,key          ;keep checking
3A085C    50        ld   a,(23560)      ;get LAST-K in A
C9        60        ret
```

Before leaving the keyboard, let's look at a couple of system variables that can be useful.

Variable	Address	Description
REPDEL	23561	Delay in 50ths of a second before the keyboard starts repeating. You can alter this value to change the delay before keyboard starts repeating
REPPER	23562	Delay between the second and subsequent key repeats. Again can be POKEd
PIP	23609	Length of click generated when the keys are pressed. Higher values give a *beep*

The other channel on an unexpanded Spectrum that is of use to us is channel P, which is used by the ZX Printer. This usually has stream 3 associated with it. Thus:

```
LD      A,3
CALL    &1601
```

will open channel P and subsequent characters will be printed on the ZX Printer if connected. It should be clear that BASIC makes wide use of the various channels: PRINT and LIST use channel S, LPRINT and LLIST use channel P and INPUT uses channel K. We can also add our own channels to the three we've discussed here, so that we can easily access additional devices such as Microdrives, different printers and other items of hardware. We'll look at this in more detail in the next instalment.

ICON TACT

You have just 100 minutes to rescue an ambassador who, with vital plans, is being held captive aboard a starship. At your disposal is a team of six individuals with various skills... A familiar scenario, perhaps, but cursor control by means of icon selection adds an exciting new dimension to Beyond Software's Shadowfire.

Shadowfire: For the Commodore 64 and Sinclair Spectrum (both versions on the same cassette) £9.95
Publishers: Beyond Software, Competition House, Farndon Road, Market Harborough, LE16 9NR
Authors: Steven Cain, Dave Colcough, Karen Davies, Graham Everett, John Gibson, Fred Gray, John Heap, Ally Noble and Colin Parrott
Joystick: Optional
Format: Cassette

Even the most casual observer will have noticed that computer games are becoming more sophisticated. Instead of slavishly reproducing arcade-style games or being content with text-only adventure games borrowed from the dungeons and dragons format, designers of home computer games are developing a style of their own. These games combine many of the features of the arcade and strategy formats to produce an entertainment which lasts considerably longer than the five minutes or so of the arcade action and is more demanding than even the brain-bending puzzles provided by the adventure.

The scenario of Shadowfire is that the renegade General Zoff has captured Ambassador Kryxix, who has the plans for a new type of spaceship called *Shadowfire*. You, the player, are charged with rescuing the ambassador before he is forced into revealing the plans, and you have 100 minutes to accomplish your task. To assist you in the rescue, you have at your disposal six characters, each with different strengths and weaknesses.

To get aboard Zoff's spaceship, your team must be 'teleported' down. The only member who can organise this is the drone Manto, so he must be sent down first to lay the teleport beam for the others to follow. However, before you dispatch them, it is advisable to arm each of the characters according to their strengths and weaknesses from the weapons available.

Shadowfire uses a unique system of moving characters, allowing them to pick up objects and to fight. Unlike other adventure-type programs that require the user to type in commands such as 'Go North' or 'Pick up laser', this game enables all functions to be performed by means of icons and a moving cursor — somewhat like the operating system used on the Apple Macintosh. To equip the team leader, Zark, with hand grenades, for example, you first select the Zark icon. Once this is chosen, the screen changes to a pictorial display of his strength, stamina and other attributes. On the right-hand side of the screen are three 'monitor' icons representing movement, battle mode and the objects screen.

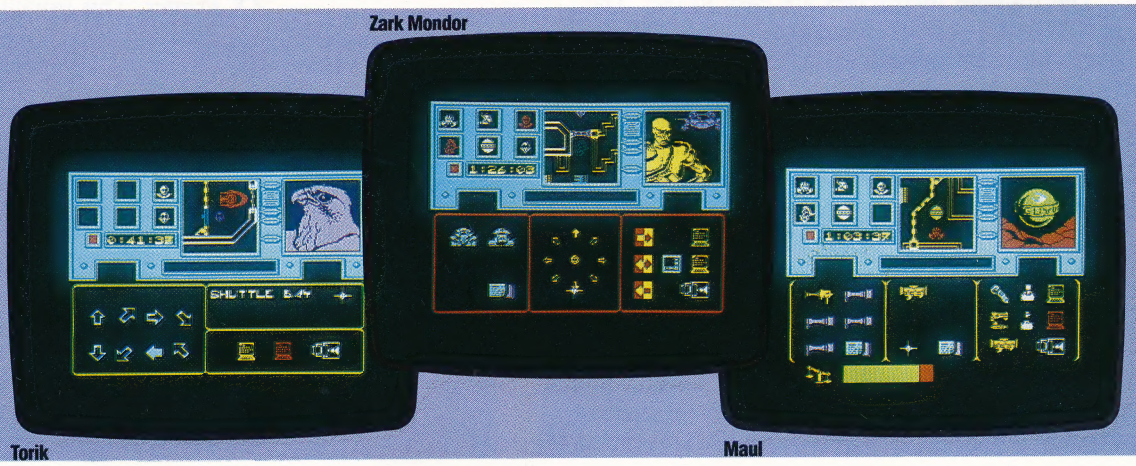
By selecting the objects screen with the cursor (which can be moved from the keyboard, joystick or light-pen), the screen changes again to display the objects that are in the same vicinity as the character as well as a number of 'activity' icons. By choosing the 'pick up' icon and then moving the cursor to the grenades icon, Zark will be equipped with grenades.

Once armed and teleported down to the spaceship, the team can begin the search for General Zoff and Ambassador Kryxix. The spaceship consists of a number of rooms and corridors, some of which contain weapons or keys to enable you to open locked doors, while others conceal enemy guards who must be destroyed before you can proceed. If you lack the appropriate key you can call upon the services of Sevrina, who can pick locks.

To make the best possible use of the 100 minutes available you should develop a strategy. Some characters are better in some situations than others and so it is advisable to have the right person in the right place at the right time.

What makes Shadowfire such an interesting game is the cursor control. The player is able to react to situations with the icon selection process much faster than he would if each command had to be typed in separately. Nevertheless, merely having fast reflexes will not allow you to complete the mission. Strategy and the ability to make quick decisions is also vital to a successful conclusion.

Players In The Drama
We show here three of the characters that the player controls in Shadowfire in different situations. On the left, Torik is in movement mode with the possible directions he can take shaded. In the centre, Zark Mondor is under attack and is in battle mode. Once again, the possible directions he can take are emphasised and his opponents are shown in icon form on the left-hand side of the screen. Maul, a battle droid, is shown with an objects screen. The centre portion at the bottom of the screen shows what objects he is currently carrying and on the left are the objects in the vicinity which can be picked up. Instructions are sent to the computer by moving a cursor (a white cross) over the selected icon and pressing the fire button on the joystick



DATABASE

Here, courtesy of Commodore Business Machines, we publish another instalment of the Commodore 64's memory map

LABEL	HEX ADDRESS	DECIMAL LOCATION	DESCRIPTION
SA	00B9	185	Current Secondary Address
FA	00BA	186	Current Device Number
FNADR	00BB–00BC	187–188	Pointer: Current File Name
ROPRTY	00BD	189	RS-232 Out Parity / Cassette Temp
FSBLK	00BE	190	Cassette Read/Write Block Count
MYCH	00BF	191	Serial Word Buffer
CAS1	00C0	192	Tape Motor Interlock
STAL	00C1–00C2	193–194	I/O Start Address
MEMUSS	00C3–00C4	195–196	Tape Load Temps
LSTX	00C5	197	Current Key Pressed: CHR$(n) 0 = No Key
NDX	00C6	198	No. of Chars. in Keyboard Buffer (Queue)
RVS	00C7	199	Flag: Print Reverse Chars.—1=Yes, 0=No Used
INDX	00C8	200	Pointer: End of Logical Line for INPUT
LXSP	00C9–00CA	201–202	Cursor X-Y Pos. at Start of INPUT
SFDX	00CB	203	Flag: Print Shifted Chars.
BLNSW	00CC	204	Cursor Blink enable: 0 = Flash Cursor
BLNCT	00CD	205	Timer: Countdown to Toggle Cursor
GDBLN	00CE	206	Character Under Cursor
BLNON	00CF	207	Flag: Last Cursor Blink On/Off
CRSW	00D0	208	Flag: INPUT or GET from Keyboard
PNT	00D1–00D2	209–210	Pointer: Current Screen Line Address

© 1983 NAKAMAE, NISHITA, OHBAYASHI—
HIROSHIMA UNIV.